IPG Politécnico da Guarda
Polytechnic of Guarda

RELATÓRIO DE PROJETO

Licenciatura em Engenharia Informática

Francisco António Brás Monteiro

dezembro | 2017

Escola Superior de Tecnologia e Gestão

Instituto Politécnico da Guarda

CLOUD-MENU

RELATÓRIO PARA A OBTENÇÃO DO GRAU DE LICENCIADO

EM ENGENHARIA INFORMÁTICA

Novembro/2017

Agradecimentos

O presente trabalho de fim de curso não poderia chegar a bom porto sem o precioso apoio de várias pessoas.

À minha esposa, Marisa, pelo incentivo, compreensão e encorajamento, durante todo este período.

À minha família, em particular, aos meus pais que sempre me incentivaram e apoiaram. E a todos aqueles com quem troquei algumas ideias sobre possíveis implementações, não podendo deixar de referir o amigo Nuno Rocha pelo apoio e paciência que teve comigo, na elaboração deste relatório.

Muito especialmente, desejo agradecer ao meu orientador professor Celestino Gonçalves, à professora Maria Clara, à professora Filipa Gaudêncio, pela disponibilidade, atenção dispensada, paciência, dedicação e profissionalismo, um muito obrigado.

Gostaria também de agradecer a todas as empresas e pessoas, que tomaram possível a elaboração este projeto, pois desenvolveram as ferramentas necessárias e as disponibilizaram de forma gratuita, para além de disponibilizarem o código fonte à comunidade.

Ficha de identificação

Francisco António Brás Monteiro

Aluno nº 1010238

Resumo

Nos últimos tempos, surgiu um novo paradigma o qual tem sido alvo de estudo académico apelidado de economia partilhada.

Os hábitos de consumo têm mudado bastante, consequência da facilidade de acesso à internet em qualquer lugar, através de dispositivos móveis que já fazem parte do quotidiano de todos nós.

Serviços como a *Airbnb* e a *Uber*, têm hoje um relevo mundial o que faz repensar os serviços originais, nestes casos específicos, o mercado de turismo e o transporte pessoal, porque os clientes cada vez mais, procuram a melhor relação qualidade/preço, tal como os produtores procuram a melhor oportunidade de negócio.

Este fenómeno leva a que produtores e consumidores convirjam para plataformas digitais, tal só é possível com o aumento da mobilidade dos equipamentos, o que permite tornar este tipo de transações comuns.

Foi neste contexto, que este trabalho foi elaborado, pretendendo facilitar a publicação do serviço de restauração, para que os clientes possam de uma forma homogénea ter acesso ao serviço de ementa do restaurante.

Palavras-chave

Economia cooperativa, *gwt*, *groovy*, xp, *horizontal scaling*

Abstract

Lately, a new paradigm has emerged which has been the target of academic study named as shared economy.

The consumption habits have changed a lot, due to the ease of access to the internet anywhere, through mobile devices that are already part of our daily lives.

Services such as Airbnb and Uber, now have a worldwide impact which rethinks the original services, in these specific cases, the tourism market and personal transport, as customers increasingly seek the best quality / price ratio, as well as producers, who's seeks the best business opportunity.

This phenomenon leads producers and consumers to converge to digital platforms, this is only possible with the increase of the devices mobility, which allows us to make this type of transactions a common habit.

It was in this context that this work was elaborated, intending to facilitate the publication of the catering service, so that customers can homogeneously have access to the restaurant's menu service.

Keywords

Cooperative economy, gwt, groovy, xp, horizontal scaling

Índice geral

Índice de ilustrações

Índice de tabelas

Lista de acrónimos

API – Application programming interface

ACID – Atomicidade, Consistência, Isolamento e Durabilidade

CDI – Context and Dependency Injection

CSS – Cascading Style Sheets

DOM – Document Object Module

DSL – Domain specific language

GPS – Global Positioning System

HTML – Hyper Text Transport Protocol

JAVA-EE – Java Enterprise Edition

JAX-RS – Java API for Restful Web Services

JVM – Java Virtual Machine

PAAS – Platform as a service

SEO - Search Engine Optimization

URL – Uniform resource locator

XML - eXtensible Markup Language

1 Introdução

Neste capítulo pretende-se abordar o tema, sem a pretensão de detalhar o trabalho em si mas, numa visão mais genérica sintetizar em que consiste o trabalho.

1.1 Motivação

Tendo em conta, que hoje em dia a maioria das pessoas já possui um *smartphone* e acesso a internet, emergiu um novo segmento da economia designada economia partilhada [1].

Este é um novo paradigma, que nasce de um novo método de consumo e do qual as tecnologias de informação são o principal veículo.

Neste contexto inserem-se aplicações como *AirBNB* e *Uber*, que pela dimensão mundial que atingiram são as mais conhecidas.

Dentro da mesma área, engloba-se esta aplicação, onde se pretende unir a vontade dos proprietários dos restaurantes de apresentarem os seus serviços e a vontade dos clientes de os consultarem e interagirem com os serviços apresentados.

1.2 Descrição do problema

O processo de gestão da maioria dos restaurantes conhecido é atualmente ainda um processo bastante manual, o que faz com que por vezes aconteçam enganos nos pedidos, atrasos nas entregas dos pedidos e confusões na ordem de atendimento. O esforço físico e o desgaste dos funcionários é bastante e faz com que a sua concentração seja afetada, motivo pelo qual foi importante pesquisar sobre uma possível solução que pudesse suprir este tipo de problema, para isso, foram criados os objetivos da aplicação como descrito no capítulo seguinte.

1.3 Objetivos

Com a ajuda desta aplicação, os utilizadores passam a ter a possibilidade de poder consultar antecipadamente os restaurantes perto de si, tal como as ementas que estes têm disponíveis. É também possível, a partir do momento em que o cliente se regista e é confirmado por um funcionário de mesa, iniciar o processo no restaurante e assim iniciar a interação com os funcionários de mesa e cozinha.

Apesar de não ter encontrado num sistema público ao nível da *cloud*, nem nos dispositivos dos próprios utilizadores, hoje em dia, o restaurante *McDonalds* permite algo deste género que é, através de um painel tátil permitir, escolher o menu, pagar e receber o pedido na mesa ou ao balcão. No próximo capítulo será apresentado um enquadramento geral do relatório de uma forma resumida.

1.4 Enquadramento

Neste capitulo é apresentado resumidamente o que será desenvolvido em cada um dos capítulos seguintes.

No segundo capítulo será apresentado o estado da arte, onde são apresentados alguns projetos que têm de alguma forma alguma funcionalidade ou na sua essência o mesmo objetivo que o pretendido no *Cloud-Menu.*

No terceiro capítulo, modelação e desenho é onde estão os diagramas que mostram os requisitos que o sistema irá necessitar além da metodologia utilizada.

No quarto capítulo é feita a referência às tecnologias necessárias para fazer a implementação em si, sendo no quinto capítulo é detalhada a forma como foi feita a implementação, fazendo referência aos componentes criados.

No sexto capitulo, para validar a implementação feita são descritos os testes unitários e de integração e a sua contextualização na integração contínua.

Por último a conclusão do relatório com as considerações para trabalho futuro.

2 Estado da Arte

Neste capítulo, através de aplicações existentes no mercado, houve a tentativa de identificar algumas funcionalidades que possam ser implementadas ou adaptadas no projeto, ou que poderão no futuro complementar as principais funcionalidades pretendidas.

2.1 Open table [2]

A aplicação *Open Table* tem como principal funcionalidade a possibilidade de ver os restaurantes que existem na base de dados, bem como o detalhe, que neste caso são o tipo de restaurante, o horário das refeições, a localização, contactos e intervalo de preços praticados.

Na aplicação é também possível fazer uma reserva, caso esteja disponível o restaurante, onde é indicado o número de pessoas para a hora pretendida, como pode ser visto na ilustração 1.

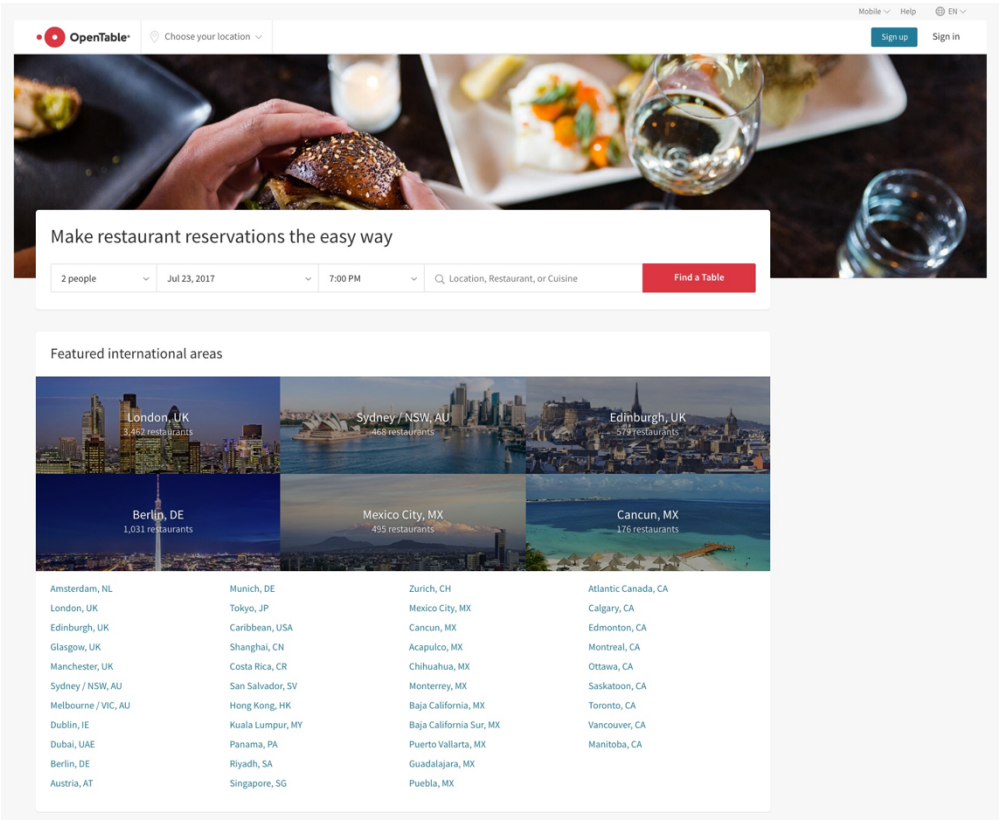


Ilustração 1: Open Table[3]

É possível também publicar a opinião sobre o restaurante e atribuir ao restaurante uma nota de zero a cinco estrelas. Esta avaliação será visível a todos os utilizadores e influenciará o *rate* do restaurante, o qual também ficara visível para o público.

Esta é uma aplicação que não tem na sua base de dados restaurantes portugueses.

Existe a versão web e mobile desta aplicação, que tem um design responsivo.

2.2 The fork [3]

Esta é uma aplicação/empresa que pertence à *TripAdvisor*, que por sinal é bastante completa.

Tem a grande vantagem de ter na sua base de dados acessível a todos restaurantes portugueses, como pode constatar na ilustração 2.

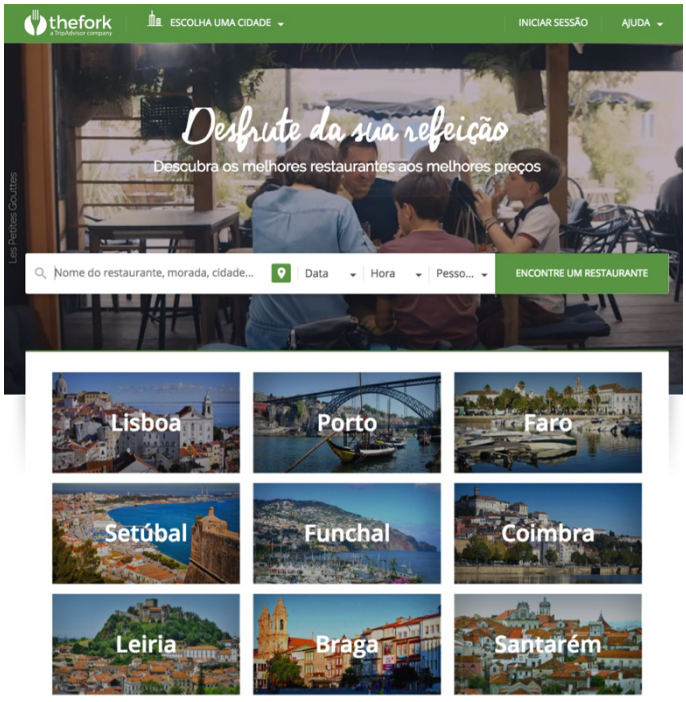


Ilustração 2: The fork [4]

É possível aqui selecionar um restaurante da base de dados da aplicação, essa seleção consegue ser bastante personalizada.

Os critérios da pesquisa podem ser o preço por pessoa, o tipo de comida, as *reviews* dos clientes, número de estrelas *Michelin*, entre outras.

É possível reservar uma mesa, ver os serviços do restaurante e o horário durante o qual o restaurante serve.

O cliente pode avaliar o restaurante num valor entre 0 e 5, avaliação esta que vai influenciar a rate visível publicamente.

Após selecionado o restaurante é possível ver o menu e os preços dos produtos.

Nesta aplicação, existe ainda um conceito de fidelização à qual a empresa apelidou de *YUM's* , essa fidelização permite descontos em restaurantes aderentes.

2.3 McDonalds

Esta é uma aplicação diferente, mas provavelmente aquela que se aproximará mais do formato que será desenvolvido neste projeto.

No interior do recinto do *McDonalds*, é possível ver dispositivos *touch* onde é possível fazer a seleção de produtos disponíveis, tal como os preços correspondentes, fazer o pagamento e optar pela entrega na mesa ou o levantamento do pedido ao balcão.

Para se ser servido à mesa, é necessário associar uma ficha numerada ao pedido feito no dispositivo *touch*, ficha essa que será colocada num lugar visível na mesa para que o funcionário possa identificar a origem do pedido efetuado.

Os projetos apresentados no estado de arte têm alguns pontos importantes como poder ver a ementa online e poder fazer um pedido no local, mas pode ser complementado com a agregação dos dois, e ainda criar uma maior interação entre os intervenientes do restaurante, no capítulo seguinte será descrita a metodologia e a modelação que foi utilizada no desenvolvimento da aplicação.

3 Desenho e modelação

Foram elaborados os diagramas com o objetivo de ajudar a perceber as funcionalidades da aplicação.

Todos os diagramas foram desenhados tendo em conta apenas a parte do *backend*, ou seja, a API pública. O que permitirá, criar outro qualquer cliente, baseando-se na documentação nos modelos mencionados.

Para a elaboração do diagrama de contexto, foi necessária alguma precisão na elaboração do mesmo, pois foi necessário efetuar a ligação dos componentes, tendo sido muito difícil encontrar um software que faça isso automaticamente a partir de um texto. Isso faz com que o seu versionamento seja mais difícil.

A ferramenta utilizada neste caso foi um software *open source* que tem um editor gratuito *online*, cuja base é a gestão de grafos, apelidada de *JGraph* [5], também esta *open source*.

3.1 Metodologia

A metodologia utilizada para desenvolver *software*, depende normalmente do contexto, da equipa e do cliente.

Os valores que são adotados nas metodologias ágeis, encaixam perfeitamente no que é pretendido para este trabalho.

Principais valores das metodologias ágeis:

- **Indivíduos e interações**, valorizando o bom ambiente entre eles e uma boa comunicação;
- **Software funcional**, em detrimento de uma documentação extensa;
- **Colaboração com o cliente final**, que neste caso será o papel do professor orientador;
- **Adaptação á mudança**, onde a criação de novos requisitos, levará a criar novas alterações na aplicação em vez de um plano rígido inicial.

Tendo em conta que será então uma metodologia ágil, a que melhor se enquadra neste contexto é a XP (*extreme programing*) [6] que pode ser explicado pelos valores que definem as principais qualidades desta metodologia:

- **Comunicação**, a comunicação entre os interessados do projeto, ajudará a ter uma visão global do projeto, facilitando assim o seu desenvolvimento;
- **Simplicidade**, o desenho da aplicação deve ser incremental, sendo que a cada iteração deve ter um desenho o mais simples possível prevendo apenas a iteração em causa;
- ***Feedback***, do sistema e da equipa:
    - ***Feedback* do sistema**, a partir da criação de testes unitários e de integração;
    - ***Feedback* da equipa,** através da definição de prioridade de implementação de requisitos e criação de testes de aceitação por parte do cliente;
- ***Coragem***, algumas atividades por parte da equipa de desenvolvimento necessitam de coragem, por exemplo a refatorização de código, por não ter um efeito imediato nas funcionalidades da aplicação;
- ***Respeito***, tendo em conta os compromissos para com o cliente, deverá ser imperativo o seu cumprimento.

3.2 Diagrama de contexto [6]

Um diagrama de contexto, permite-nos ver como é que o sistema interage com o seu ambiente, o que nos dá uma visão global do funcionamento, como se verifica na ilustração 3.

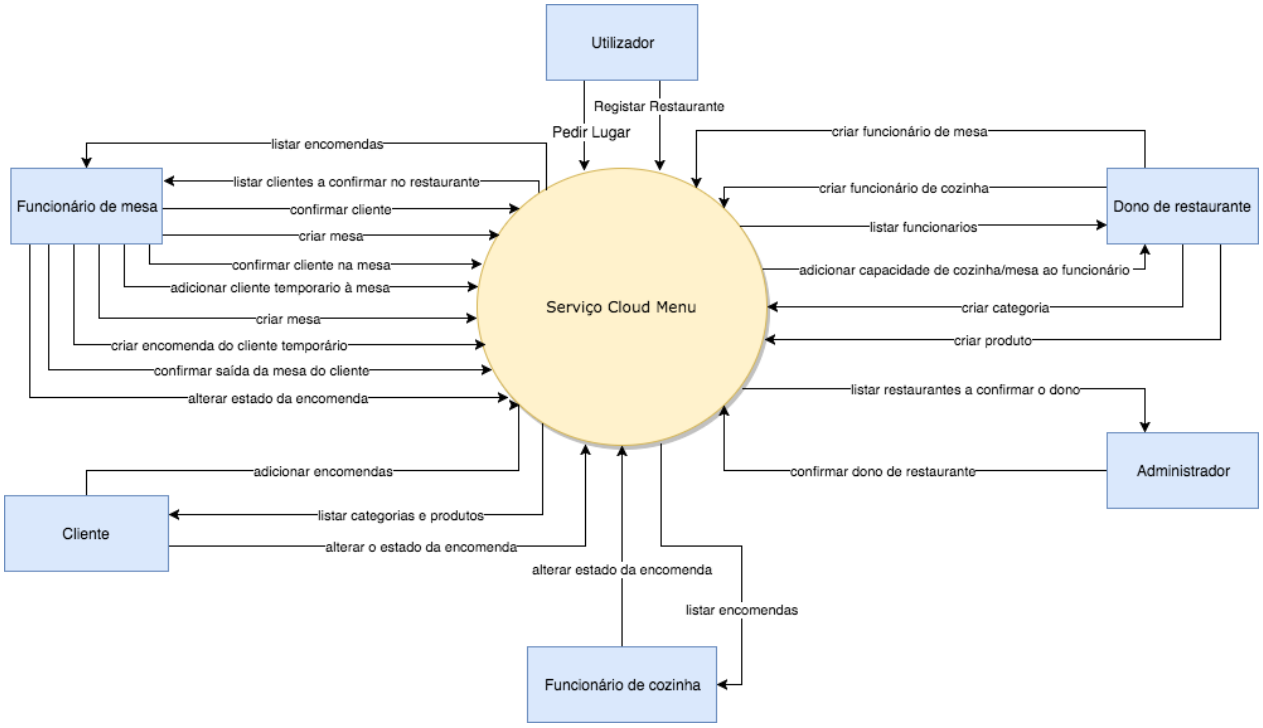


Ilustração 3:Diagrama de contexto

3.3 Entidades

Sendo este um modelo de dados não relacional, mas ainda assim sendo importante a construção de um modelo graficamente, foi tomada a decisão de criar o modelo de classes que é utilizado para fazer a persistência no *datastore*.

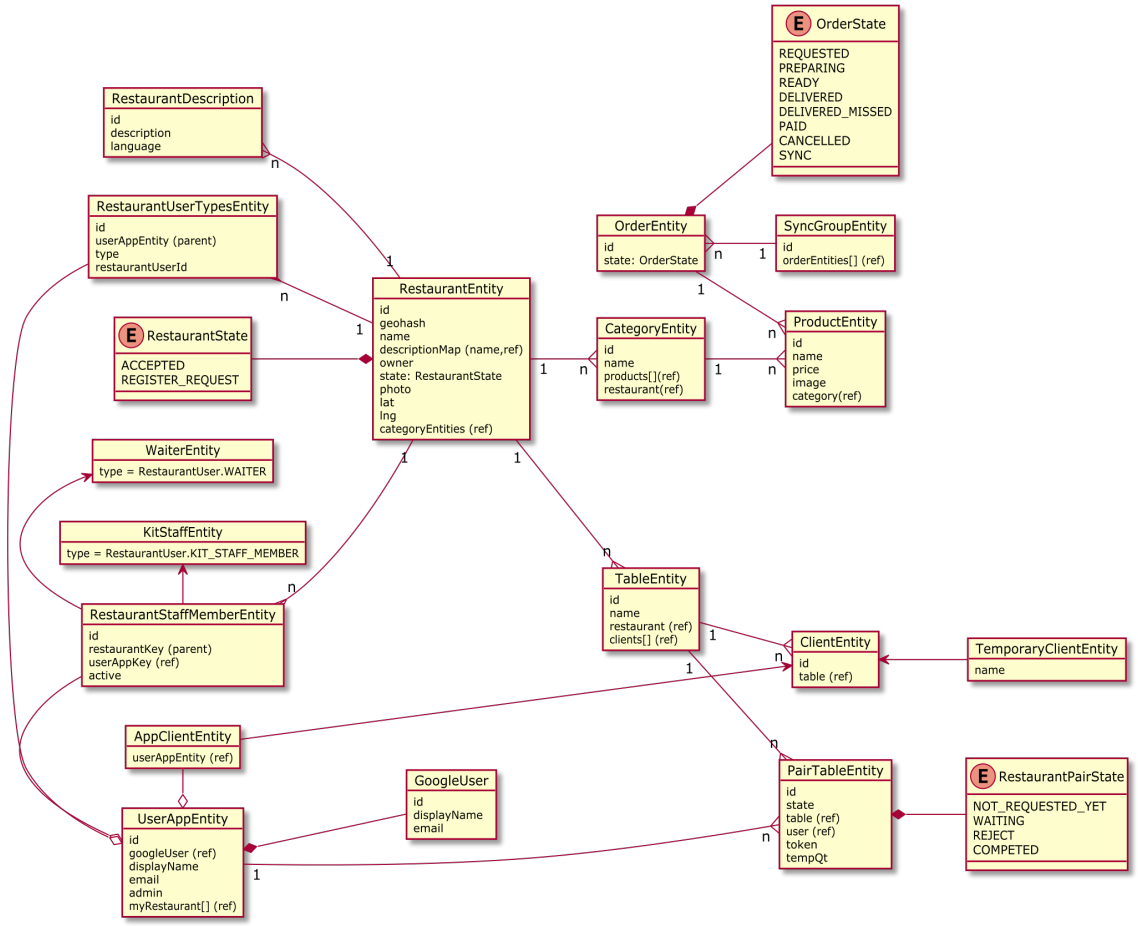


Ilustração 4: Entidades

3.4 Dicionário de dados

Foi usada uma *framework* para a persistência de dados que utiliza as classes na aplicação para esse propósito, as quais denominamos de entidades. Através de um mecanismo de serialização dos dados, os valores contidos nas instâncias, são persistidos através do *Objetify* no *Datastore*. O mesmo acontece na leitura, mas no sentido inverso, ou seja, após a leitura feita no *Datastore* o *Objetify* retorna as entidades correspondentes. Tendo em conta que não se trata de uma base de dados relacional, e as restrições são aplicadas ao nível do servidor aplicacional, foi necessário fazer algumas adaptações ao modelo tradicional do dicionário de dados, a coluna restrição deixa de fazer sentido, por esse motivo é excluída, além disso existem diferentes tipos de dados que por norma não são utilizados no modelo tradicional.

Tipos de dados como as coleções, os enumerados e as referências, têm aqui algum destaque, uma vez que são usados apenas neste contexto de base de dados não relacional.

Os enumerados são tipos de dados que têm uma lista constante de valores os quais de alguma forma descrevem o valor em si, como é o caso dos estados.

As referências são representações de um registo em particular de uma entidade e são semelhantes nas bases de dados relacionais, a uma chave estrangeira.

As coleções são também especiais e estas podem ser mapas, listas ou *sets*.

Os mapas são pares de associações do tipo chave, valor onde os valores podem ser repetidos, ao contrário das chaves que são únicas.

As listas são coleções de valores que podem ser repetidos.

Os *sets* são coleções onde todos os seus valores são únicos.

O dicionário de dados da entidade *RestauranteEntity* pode ser visto na tabela 1.

RestauranteEntity

Tabela 1:Dicionário de dados de RestaurantEntity

Nome do campo	**Tipo de dados**	**Descrição**
geohash	String	Hash de localização
name	String	nome
descriptionMap	Map	Mapa com uma string que representará o código do país como por exemplo “PT” e a referência na entidade RestaurantDescription
owner	Key	Referência na entidade UserAppEntity contendo os dados do proprietário do restaurante
state	RestaurantState	Enumeração com a lista
photo	String	Caminho para a imagem guardada no *CloudStorage*
lat	Double	Latitude
lng	Double	Longitude
categoryEntities	Set	Set de referencias da entidade CategoryEntity

As restantes tabelas do dicionário de dados podem ser encontradas em anexo.

3.5 Diagramas de sequência

Os diagramas de sequência permitem analisar a sequência da interação entre os atores e a ação em si.

O diagrama de sequência da criação de um restaurante pode ser visto na ilustração 5, os restantes podem ser vistos em anexo.

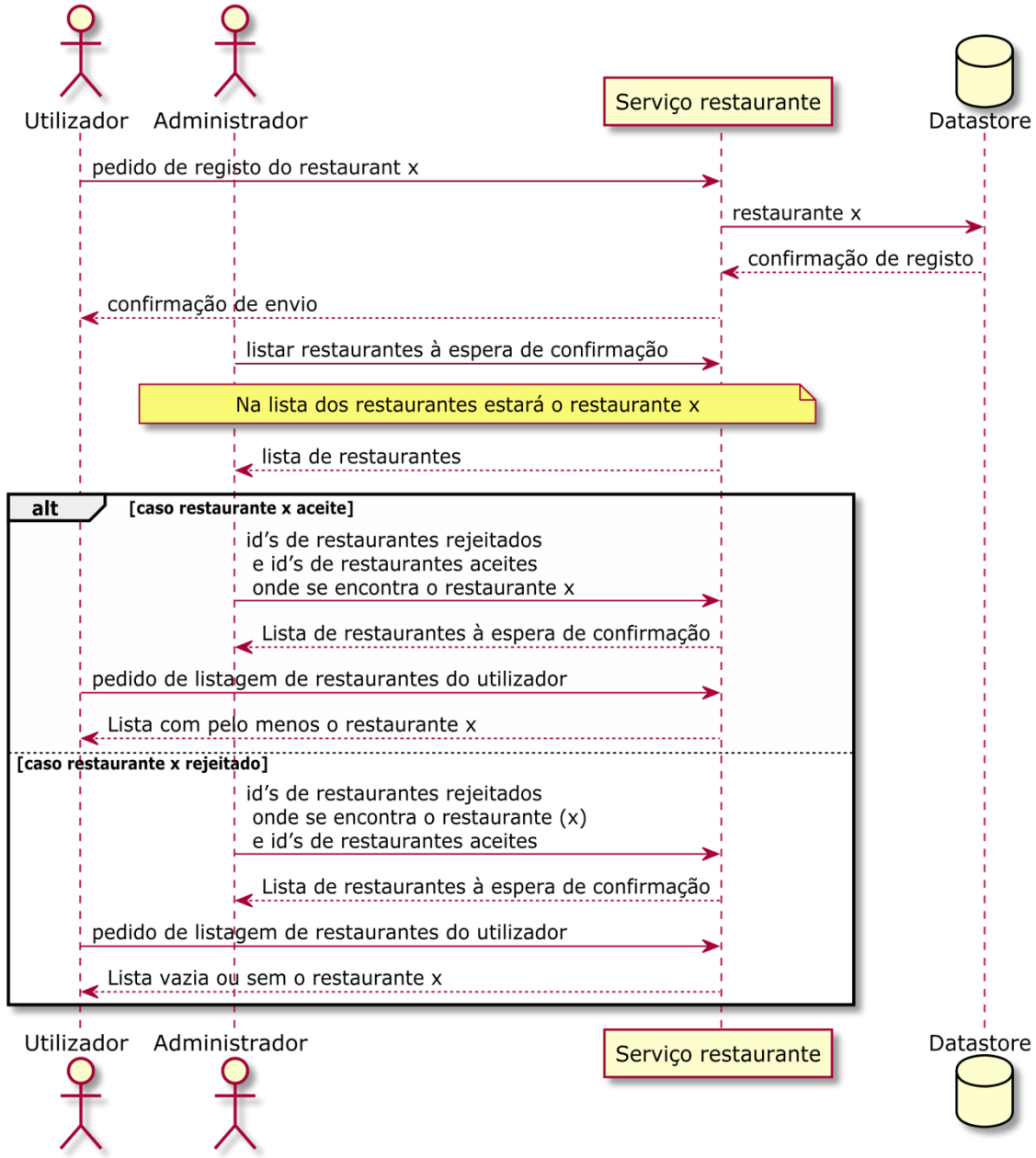


Ilustração 5:Diagrama de sequência para criar restaurante

3.6 Diagrama de estados

Para ser mais preciso em relação às alterações do estado dos pedidos, alterações essas feitas pelo funcionário de cozinha, funcionário de mesa ou o cliente, foi adicionado um diagrama de estados, onde constam as alterações dos estados e os atores autorizados a cada alteração, bem como o cliente.

O diagrama de estados dos estados da encomenda, podem ser vistos respetivamente na ilustração 6, onde a abreviatura F.C. corresponderá a funcionário de cozinha e F.M. a funcionário de mesa.

Ilustração 6: Diagrama de estados da encomenda

Para a elaboração dos diagramas de sequência e do diagrama de estados foi utilizada a ferramenta *Asciidoctor* com o *plugin Asciidoctor-diagrams*. [7]

Esta ferramenta permite gerar páginas web estáticas a partir de uma linguagem de texto simples, acrescentado apenas alguns carateres especiais e delimitando com blocos específicos.

O *plugin* de diagramas dá acesso a várias ferramentas de renderização de diagramas das quais foi usada para este trabalho o *PlantUML* [8].

Sendo este tipo de diagramas essenciais para perceber o que implementar e como, faz sentido incluir-se no projeto estes diagramas, sendo versionado com o próprio código fonte, entrando no mesmo ciclo de vida que os restantes recursos da aplicação.

Tanto a modelação como a implementação da aplicação, levaram à escolha de tecnologias apropriadas, as quais serão descritas no capítulo seguinte.

4 Tecnologias

Neste capítulo, pretende-se detalhar a utilização das ferramentas utilizadas, de forma a obter os resultados pretendidos. Tendo em conta as possibilidades disponíveis no mercado, e mostrando também outras possíveis alternativas, fundamentando o mais possível a escolha em causa.

Este trabalho, tal como o próprio nome sugere, foi alojado na *cloud*. Existem várias formas de disponibilização de serviços na cloud, uma delas é o PAAS, que consiste em disponibilizar uma plataforma completa de serviços, não tendo o programador que se preocupar com a sua gestão, como o *load balancing* de serviços, etc.

A PAAS para o alojamento da aplicação foi a *appengine*. Esta plataforma existe com algumas variantes, tanto a nível de linguagens de programação, como de ferramentas de base de dados. As linguagens de programação utilizadas pelo autor, foram o *java*, e o *groovy* em contextos similares, mas também diferentes.

4.1 *Java*

A linguagem *java* foi a predominante no projeto, é a linguagem de eleição do autor por ter alguma experiência profissional no seu quotidiano. Trata-se de uma linguagem orientada a objetos, a qual permite, neste contexto, construir o código parcialmente nos três módulos no projeto. São estes módulos:

- O *backend*, o qual tem suporte à versão 1.7 do *java*;
- O *frontend*, que é um módulo que tem atualmente suporte para a versão 1.8 do *java*;
- O módulo partilhado, por ser uma dependência do modulo de *backend* foi necessário ser configurado como uma biblioteca de suporte à versão 1.7

4.2 *Groovy*

Esta é uma linguagem que partilha muito com a linguagem *java*, é uma linguagem que funciona na JVM, como o *java*, em particular no módulo de *backend*. Foi escolhido o *groovy* como linguagem complementar pois no *backend*, por exemplo, não é possível usar

lambdas com *java* 1.7. (versão que foi usada no módulo de *backend*) que são bastante úteis para a manipulação de coleções, filtros etc.

Embora também não existam no *groovy,* os *lambdas*, existe uma forma equivalente de fazer esse tipo de tratamentos, através de *closures*. As diferenças entre um lambda e uma *closure* não são grandes, um *lambda* é apenas uma função *inline*, ou seja, não podendo capturar variáveis, pode atribuir-se essa função a uma variável local, de instância, ou a um parâmetro de um método no caso do *java*. Para se poder passar para o interior de um lambda uma variável em *java*, imaginando este cenário dentro de um método, é por exemplo necessário que essa referência tenha o equivalente a um modificador final ou seja, após a sua criação não pode ser alterado [9]

As *closures*, em *groovy* têm também este comportamento, mas é possível passar referências que assumam valores variáveis. Além dessas características, as *closures* em *groovy* permitem criar um parâmetro por defeito, que quando não é feito explicitamente, será denominado *it*. Além destas vantagens o *groovy* permite entre outras coisas abdicar da criação de *getters* e *setters* explicitamente, uma vez que estes são chamados com a definição de atributos do *groovy*. Para uma melhor perceção, abaixo é apresentada a tabela 3 da chamada de propriedades em java e em *groovy*.

Tabela 2: exemplo de diferenças entre java e groovy

Java	**Groovy**
exemplo.setValor(“1”)	exemplo.valor = ‘1’
System.out.println(exemplo.getValor())	println exemplo.valor

Outra utilização do *groovy* é a possibilidade de ser utilizado como *script*, sendo que é possível injetar variáveis na sua chamada, que em combinação com os *closures* da linguagem faz desta técnica uma ferramenta ótima para criar DSL’s (*Domain Specific Language*). Um dos exemplos de DSL*’s groovy* são os *scripts* de *build* da ferramenta *gradle* a qual é usada para gerir a *build* deste projeto.

Apesar do módulo de *frontend* ter sido feito maioritariamente em *java*, este na compilação será transformado em código fonte *javascript*, que é a linguagem de programação que o *browser* reconhece. Como há casos em que é necessário integrar bibliotecas *javacript* externas no módulo cliente, (a descrição dos módulos será feita no capítulo de

desenvolvimento), é necessário algum código *javascript* nativo para integrar estes componentes externos.

4.3 HTML, DOM e técnicas de renderização

HTML [9] é uma linguagem de marcação, que serve para construir uma árvore em memória, na qual os nós conterão propriedades com informação importante para a renderização que o *browser* deverá efetuar.

Está árvore denomina-se DOM [10] e pode ser lida pelo browser de forma declarativa na forma de *html,* ou imperativa, na forma de *javascript*

4.4 Renderização do lado do servidor

A renderização do lado do servidor é a forma mais tradicional de criar conteúdo HTML dinâmico. Trata-se de criar dinamicamente as *strings* que formarão o HTML que será lido no browser. No caso deste projeto, trata-se apenas da página inicial que conterá apenas as dependências *javascript* e do *body*.

4.5 Renderização do lado do cliente

Esta renderização, que é feita após a parte da receção do HTML por parte do cliente, que como referido anteriormente, são apenas os *scripts* no *header* e o *body*. Estes *scripts* farão com que, após o elemento *body* ser renderizado, adicione à DOM os restantes elementos que constituirão a aplicação, dependendo do URL em causa. Esta é uma técnica que é muito usada em aplicações do tipo *Single Page Application,* uma vez que, no caso de transições entre *views* pode ser feito apenas recebendo os dados em *json* e aplicando uma função do lado do cliente para ser feita a aplicação do lado do cliente.

Deste modo, o primeiro carregamento será mais pesado, porque será necessário descarregar todos os *scripts* necessários para as alterações na DOM através da interação com o utilizador. Da próxima vez que o utilizador tentar aceder à página, o conteúdo deverá estar em *cache* o que fará com que a aplicação seja mais rápida, tendo em conta que na rede apenas serão transmitidos dados relativos ao modelo de dados e não as *views*

e o HTML associado. Mas nem tudo são vantagens, em relação à renderização do lado do cliente.

Normalmente os algoritmos de classificação de páginas utilizam *bots* e não é conhecida a profundidade da análise que estes fazem no *javascript,* isto no caso de analisarem também o *javascript*. Tendo isso em conta, uma página com o mesmo conteúdo, será mais provável ter num *rank* superior quando é uma página com a DOM construída estaticamente, ou seja, contruída no lado do servidor. A este tipo de técnicas, denomina-se SEO, no âmbito do projeto, não foi tido em conta.

4.6 Ferramentas de *build*

Para fazer a integração entre os módulos, foi necessário recorrer a uma ferramenta de *build*. Entre as ferramentas existentes, foram consideradas como candidatas o *maven* e o *gradle*. O *maven,* por ser a ferramenta com mais anos de utilização no mercado empresarial, ou seja, mais madura, as características desta ferramenta de *build* são: capacidade de a partir de um padrão de projeto, características como a pasta do código fonte, pasta de testes unitários, e de *resources* sejam sempre em caminhos padrão. Acesso a *plugins* para execução de várias tarefas necessárias, como criação de pacotes para *deploy*, compilação de *groovy*, gestão de dependências internas e externas, entre outras capacidades.

O *gradle* também foi eleita por ser uma ferramenta mais recente, e com particularidades não presentes no *maven*, mas como todas as características apontadas anteriormente para o caso do *maven* se aplicam ao caso *gradle,* com uma grande diferença entre os dois, o *maven* é uma ferramenta declarativa, o que significa que os *scripts* de *build* no caso do *maven* são feitos de forma declarativa, mais concretamente, são *scripts* XML. Em cada módulo *maven*, existirá um ficheiro *pom.xml* que indicará como será feita a *build* recursivamente, pois os módulos estarão organizados de forma hierárquica. No caso da construção dos *scripts gradle,* esta é feita de forma imperativa. Os *scripts*, neste caso são DSL's *groovy* criados a partir do *gradle,* no qual são injetadas as variáveis de contexto, para em conjunto com o desenvolvimento que o programador acrescentar poder executar a *build.*

4.7 Versionamento do código e da documentação

A necessidade de versionamento do conteúdo de um projeto é essencial e de extrema importância no mundo do desenvolvimento de *software*. Dificilmente se consegue produzir *software* bem à primeira tentativa, principalmente quando não temos experiência nas tecnologias que estamos a utilizar. Mas facilmente conseguimos definir metas parciais que nos levem a um ponto em que possamos dizer que a etapa está completa e é aceitável. Ao definirmos uma meta, após alcançarmos o aceitável podemos marcar um ponto de regresso, caso alguma coisa corra mal. Este é o tipo de benefício, que é encontrado quando é apenas um *developer* a desenvolver.

No caso de ser uma equipa, é ainda mais vantajoso, uma vez que é possível até alterar o mesmo ficheiro e após enviar para o servidor ser possível uma junção automática dos blocos feitos por ambos, leia-se *merge*, no caso de ser em linhas diferentes.

O versionamento do código foi feito na plataforma *Bitbucket*. Esta plataforma utiliza a ferramenta *GIT* para o versionamento do código e possibilita a criação de um repositório privado gratuitamente [9].

4.8 *Appengine Java Server*

Para servir os pedidos vindos dos *web-clients,* foi utilizado um servidor proprietário da Google para a plataforma *AppEngine*. Apesar de não ser um servidor *full-stack* é um servidor extremamente rápido a iniciar, o que é imperativo numa arquitetura de micro serviços, preparada para um crescimento horizontal [10], tema que será abordado no capítulo do desenvolvimento.

4.9 *Datastore*

O *Google AppEngine*, plataforma utilizada para o desenvolvimento tem duas principais opções de base de dados, a relacional e *NOSQL*. Como todas as escolhas recaíram na alta escalabilidade, o *Datastore* foi a escolha de eleição, não tendo o suporte a ACID das bases de dados relacionais, é possível mesmo assim ter acesso a transações, o que permite atualizar dados em segurança. É frequente nas bases de dados relacionais o *scaling* ser feito através da criação de novos servidores de leitura, para a base de dados que terá um único servidor de escrita.

No caso do *Datastore* e de outras bases de dados *NOSQL,* não é aconselhado num cenário de produção existir apenas um nó, porque é provável que não esteja otimizado para esse fim, mas sim para trabalhar com um *cluster* ou mais.

Os *clusters* utilizados são tolerantes a falhas, o que quer dizer que não é problemático, se um nó deixar de funcionar, bastando para isso colocar outro em funcionamento.

Para demonstração da aplicação, devido aos custos associados, foi apenas utilizado um nó, embora quando a aplicação estiver oficialmente em produção seja desaconselhado a permanência com um só nó, por questões de performance e de disponibilidade.

4.10 *Cloud Storage* [11]

Este é o tipo de base de dados, com dados não indexados, a não ser pelo nome do objeto.

É utilizada para guardar dinamicamente ficheiros de grande porte, por exemplo, imagens, vídeo etc.

4.11 *OAuth2 – Google plus* [12]

O serviço de autenticação foi feito no *Google Plus*, com o protocolo *OAuth2,* o que permite a autenticação de utilizadores por entidades externas.

4.12 *Jersey* [13]

Framwork rest que implementa a especificação do *JAX-RS*, especificação do *JAVA-EE* para servir *web services* do tipo *rest*, o *google app engine* tem incluída uma implementação de *web-services* rest, contudo, foi esta a escolhido, o *Jersey* por ser mais fácil integrar com o *RestyGWT*.

4.13 *Guice* [14]

Esta é uma ferramenta de injeção de dependências, padrão de desenvolvimento, que torna baixo o vínculo entre objetos.

Com um baixo vinculo torna-se prático substituir objetos nos quais é preciso simular comportamentos, no caso de testes por exemplo.

Também foi considerado o *CDI,* que é a *framework* de injeção de dependências do *JAVA-EE*, contudo a configuração do *Guice* é sem XML, ou seja, apenas *java*, razão pela qual foi a escolhida.

4.14 *Swagger* [15]

Esta é uma *framework* para criar contratos nos *web-services*, utiliza um ficheiro *JSON* com a descrição dos serviços onde constam o URL, o caminho, o método utilizado para fazer o pedido, os tipos de entrada e o tipo de saída. Os tipos são descritos na forma de *json-schema* que é o padrão de representação e validação de tipos *JSON*.

Utilizando esta ferramenta, é possível gerar clientes em várias linguagens e construir automaticamente uma interface padrão que permite invocar os *web-services* a partir do *browser*.

4.15 *GWT* [16]

Gwt é uma ferramenta que permite criar código em *java*, usar padrões que os programadores estão habituados da plataforma *java* (como por exemplo, recursos de internacionalização) para programar o código que vai ser lido pelo *browser.*

Com esta ferramenta, é possível utilizar bibliotecas externas desenvolvidas em *java*, com a ressalva de que estas dependências precisam ter o código fonte incluído.

O código *java* após compilado, será neste caso *JavaScript* e não *bycode*, pois será utilizado um compilador *GWT* a partir do *plugin* do *Gradle.*

4.16 *GwtBootstrap3* [17]

Biblioteca com componentes gráficos para utilização em *GWT*, o nome é auto descritivo, tem como principal objetivo, utilizar a biblioteca de componentes do *bootstrap* que é uma biblioteca composta por HTML, CSS e *javascript*.

Esta biblioteca implementando os padrões do GWT permite algumas funcionalidades que merecem ser referidas:

- O padrão *editor* permite atualizar um objeto em *java* a partir de um componente visual e vice-versa;
- A gestão de erros pode ser feita no GWT com *validators*. Estas classes implementam a interface *Validator* a qual terá a lógica que valida o conteúdo do componente visual quando é chamado num formulário, o erro dará lugar a uma mensagem visual;
- Outra forma de fazer a validação é a utilização de anotações nos campos do objeto, como está descrito na especificação *java JSR-303;*
- Como exemplo, a anotação *@NotNull,* colocada acima da declaração de um campo, valida que o campo não pode ser vazio.

4.17 *RestyGwt* [18]

Uma das grandes vantagens de usar *java* do lado do cliente, é que é possível partilhar código entre este e o servidor. Como esta ferramenta reutiliza parte da especificação do *JAX-RS*, é uma grande valia na reutilização de código. Como por exemplo, as mesmas anotações que são usadas no servidor, nos métodos publicados, são usadas no cliente nos métodos consumidos.

4.18 *TestNG* [19]

Framwork de testes unitários semelhante ao *JUnit*. Através da anotação *@Test,* tanto o *JUnit* como o *TestNG*, declaram que o método da *class* anotado irá executar o seu conteúdo que será avaliado como *passed* ou *fail*.

Embora sendo o *JUnit*, a ferramenta de testes mais utilizada em combinação com a linguagem *java*, a opção eleita foi o *TestNG* porque, permite uma maior personalização.

Possibilidades como dependência entre testes e a possibilidade de correr testes em paralelo, fez do *TestNG* a ferramenta de eleição, porque com este tipo de possibilidades a execução de testes torna-se mais rápida.

5 Implementação

Este capítulo, pretende demostrar algumas das opções tomadas na forma da implementação e os pressupostos que a isso levaram, além de explanar o mais possível o funcionamento de cada parte identificável da aplicação.

5.1 Encaminhamento de URL's para renderização de componentes

Uma aplicação *web* do tipo *single page aplication,* como o próprio nome indica tem apenas uma página. Para a implementação do projeto foram criados painéis mapeados com âncoras no URL [1].

Onde https://cloud-restaurant-menu.appspot.com/ é a parte comum do URL, a partir do # *(hash,* que neste caso representa a âncora), que é inserida a parte do serviço em si.

Através desta técnica é possível mudar o URL no *browser,* sem que seja feita uma chamada ao servidor.

Por outro lado, através da mudança de URL no *browser*, possibilita guardar no histórico da navegação. Uma vez que o conteúdo estático é guardado em *cache*, (o que é o caso do *JavaScript* proveniente do *GWT)* na exceção do primeiro carregamento, a aplicação, ficará muito mais rápida no que diz respeito à renderização de conteúdos.

[1] https://cloud-restaurant-menu.appspot.com/#restaurant/5154545407623168

5.2 Hierarquias de componentes

Nesta secção, serão descritos os componentes renderizados, tendo em conta o URL do *browser* e o contexto do utilizador.

O primeiro nível de agrupamento de componentes será o contentor raiz, que é constituído por um contentor superior e um inferior.

No contentor superior, ficam os menus do utilizador, os quais dependem do utilizador em causa, o *link* para voltar à pagina inicial, (no caso de ser um cliente e ter pedidos em curso como no exemplo na ilustração abaixo com uma encomenda em curso) um botão para mostrar um painel com os pedidos em curso (como pode ser visto na ilustração 7) e o *link* de *login*/*logout.*

Ilustração 7: botão de acesso a pedidos em curso

No contentor inferior fica o conteúdo da aplicação correspondente ao contexto do URL.

Cada utilizador terá os menus apresentados na tabela 4, podendo ainda ser acumulados no caso do proprietário, uma vez que também é um utilizador.

Tabela 3: Menus por utilizador

Utilizador (não autenticado)	(Sem menus)
Utilizador autenticado	Criar restaurante
Proprietário	Meus restaurantes
Administrador	Restaurantes a confirmar

5.3 Descrição de páginas e separadores

Para simplificação da apresentação, os caminhos serão representados com a notação caminho/{variável} onde as variáveis serão descritas.

5.3.1 Home:

Na página inicial que terá o componente *Google Maps*, irão ser apresentados os restaurantes próximos da localização, que o utilizador indica na caixa de texto contida no mapa. Esta página será vista por todos os utilizadores, da mesma forma.

Caminho: https://cloud-restaurant-menu.appspot.com/#

5.3.2 Criar restaurante:

Página onde um utilizador pode criar um restaurante, onde será mostrado um formulário com os dados do restaurante, o nome, a descrição e as coordenadas GPS [20] que são capturadas a partir do mapa (este componente será detalhado no final do capítulo) e a imagem do restaurante.

Um dos campos registados, o GPS, descrito de uma forma simplista é obtido através do posicionamento de satélites.

Esta página só pode ser acedida por utilizadores autenticados.

Caminho: https://cloud-restaurant-menu.appspot.com/#res-reg

5.3.3 Restaurantes a confirmar:

Página onde o administrador deve confirmar os restaurantes, para que possam ser visualizados pelos utilizadores.

Esta página só pode ser acedida por utilizadores com o perfil de administrador.

Caminho: https://cloud-restaurant-menu.appspot.com/#rest-reg-req-list

5.3.4 Meus Restaurantes

Nesta página é possível ver todos os restaurantes onde o utilizador se registou. Surgirá um componente por restaurante inserido, onde é identificado o nome do mesmo, o estado de validação e a descrição do restaurante.

Caso o estado do restaurante seja **validado**, o título do cabeçalho terá o *link* para o restaurante.

Como a própria funcionalidade indica, esta página estará disponível apenas para os proprietários de um restaurante.

Caminho: https://cloud-restaurant-menu.appspot.com/#my-restaurants

5.3.5 Página do restaurante

Esta é a página que merece um maior destaque, pois é aqui que se centra a maior parte das funcionalidades da aplicação, pois é onde se encontra todo o contexto do restaurante selecionado.

Para distribuir as funcionalidades, esta página foi dividida em separadores e para simplificar a sua apresentação, serão descritas com as funcionalidades apresentadas.

Esta também é a página que apresenta mais diferenças em relação ao conteúdo apresentado por utilizador, pois o tipo de utilizador influencia o conteúdo e o funcionamento dos separadores.

Esta página apresentará o nome do restaurante e o contentor de separadores.

Caminho: https://cloud-restaurant-menu.appspot.com/#restaurant/{id do restaurante}

5.3.6 Página do restaurante – Separador (Menu)

Neste separador é mostrada uma lista de painéis, onde cada um representa uma categoria de produtos, cada painel contém um título no cabeçalho e uma lista de produtos.

O comportamento deste painel difere dependendo da autorização do utilizador.

Apenas os donos do restaurante podem gerir categorias, isto é, criar, remover, alterar nome, adicionar e remover artigos. Todos os outros utilizadores terão apenas o acesso de leitura ao componente.

O componente artigo, também é um componente dependente do perfil de utilizador.

Caso seja dono no restaurante e o componente da categoria esteja em modo de edição o utilizador tem a possibilidade de alterar o nome, a imagem e o preço do artigo.

Caso seja um cliente, ou seja, um utilizador autenticado, (que é emparelhado por um empregado de mesa do restaurante) terá um *link* que após clicado, dará lugar a uma caixa de texto para introduzir a quantidade, um *link* para cancelar o pedido do artigo e um botão como visto na ilustração 8.

Ilustração 8: Formulário para criação de pedido

Após ser criado o pedido, ficará num componente no cabeçalho, como mencionado no capítulo de hierarquia de componentes, que terá um aspeto como mostrado na ilustração 9.

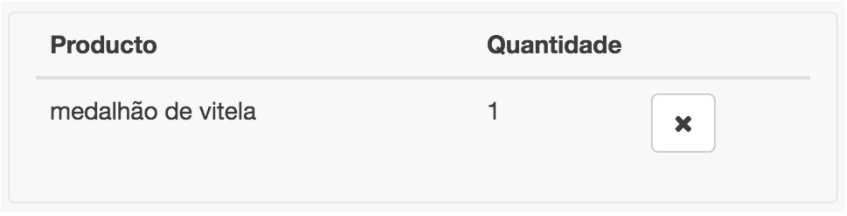


Ilustração 9:Pedidos não confirmados

Para os restantes utilizadores, o componente será apenas de leitura e mostrará o preço e o nome, além disso também um botão para poder eliminar o artigo que pediu anteriormente.

5.3.7 Página do restaurante – Separador (Funcionários)

Esta separador pretende gerir os funcionários associados ao restaurante, dividida em duas zonas. A zona da **inserção de funcionários** e a zona de **funcionários ativos**.

Na zona de inserção de funcionários é possível:

- A representação de cada funcionário neste formulário, é feito a partir do par de componentes **tipo** e ***e-mail***;
- O tipo é uma caixa de seleção onde é possível selecionar uma de duas opções, empregado de cozinha ou empregado de mesa;
- Sendo que o *e-mail* é o *e-mail* associado à conta Google, que está por isso associado à sua autenticação.

Este par de componentes, tem uma quantidade variável, podendo ser adicionado no botão **adicionar funcionário** e removido clicando na cruz posicionada no componente *e-mail*. Para confirmar a inserção de todos os *e-mails*, como funcionários bastará clicar em **confirmar novos funcionários**.

Na zona de funcionários ativos é possível:

Ativar e desativar cada funcionário com o tipo correspondente.

Os funcionários só poderão ter apenas um tipo ativo, mas podem ter os dois tipos registados.

A este separador, só tem acesso o proprietário do restaurante.

5.3.8 Página do restaurante – Separador (Pedir Lugar)

Este é o separador, onde o utilizador pode pedir a confirmação da presença no restaurante.

Existe na aplicação um conceito que foi aplicado aos clientes que não usam a aplicação, ao longo deste relatório serão denominados como clientes temporários.

O cliente pode então escolher a mesa no componente de seleção de mesas e registar no formulário o número de clientes temporários.

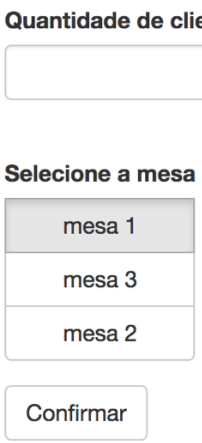


Ilustração 10:Formulário para a inserção de pedido de mesa

Após o utilizador clicar no botão confirmar, os componentes mencionados darão lugar apenas ao código registado, o qual, fisicamente será mostrado ao funcionário de mesa que fará a confirmação visual do código na sua aplicação, como pode ser visto na ilustração 11.

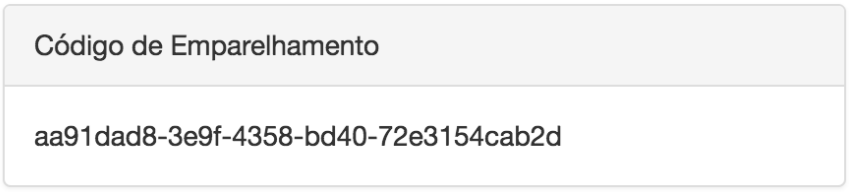


Ilustração 11: Exemplo de código de emparelhamento

A este processo denominamos emparelhamento. Este separador é acessível a todos os utilizadores que não sejam funcionários.

5.3.9 Página do restaurante – Separador (Mesas)

Este é um separador que tem uma apresentação diferente, dependente do utilizador que a acede.

No caso de ser o proprietário do restaurante, na zona de **ocupação de mesas e pedidos,** é possível ver todas as mesas que existem no restaurante.

Cada componente da mesa, é um painel que tem no canto superior direito uma cruz onde é possível eliminar a mesa, caso esta não tenha clientes e o utilizador seja o proprietário do restaurante, como pode se visto na ilustração 12.

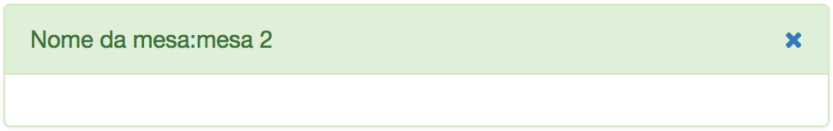


Ilustração 12: Componente mesa que pode ser eliminada por pelo proprietário do restaurante

Se o utilizador for um funcionário de mesa, no mesmo sitio aparecerá um *icon* de um utilizador com um pequeno mais à sua direita. Este *icon* permitirá adicionar um cliente temporário na mesa respetiva como pode ser visto na ilustração 13.

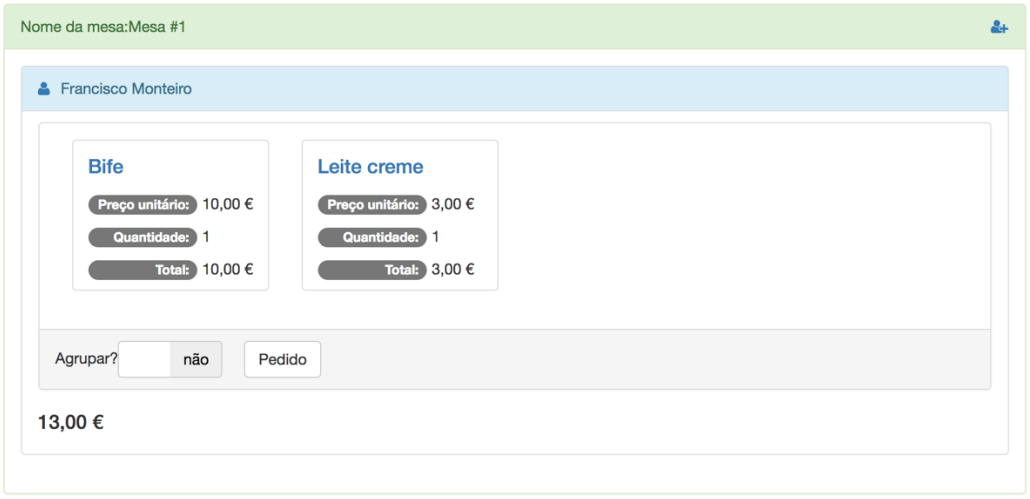


Ilustração 13: Componente da mesa e total a pagar pelo cliente

Ainda dentro da componente de mesa temos os componentes de clientes.

O componente do cliente, difere ligeiramente entre os perfis que o podem visualizar que são neste caso, os funcionários e o proprietário do restaurante, mas todos podem ver os pedidos, o estado dos pedidos e o total a pagar pelo cliente.

No caso dos funcionários, o componente de estado da encomenda, será uma caixa de seleção, onde será possível selecionar um estado seguinte, no caso de ter autorização para transitar para um novo estado, caso contrário será apresentado apenas o texto do estado.

O funcionário pode ainda confirmar a saída de um cliente, a partir de um *icon* que é uma seta horizontal para a direita, no canto superior direito do componente do cliente, caso todos os pedidos que o cliente realize estejam num estado final.

O proprietário tem também acesso à zona de novas mesas, onde é possível adicionar uma mesa, colocando o nome na caixa de texto e clicando em adicionar mesa.

5.3.10 Página do restaurante – Separador (Confirmação Presença)

Esta separador é apenas acessível ao funcionário de mesa do restaurante e está relacionada com o separador pedir lugar mencionada no capítulo do Separador (Pedir Lugar).

Aqui serão listados os utilizadores que fizeram pedidos, para serem vinculados a uma mesa como o exemplo visto na ilustração 14.

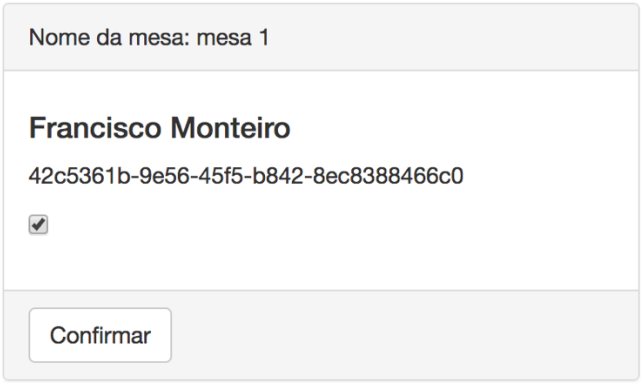


Ilustração 14: Confirmação visual de cliente

Após confirmar na caixa de verificação e clicar no botão **Confirmar**, fará com que o utilizador que fez o pedido passe a cliente associado à mesa que selecionou do restaurante, como no caso do exemplo à mesa 1.

5.3.11 Página do restaurante – Separador (Pedidos)

Este separador é composto por dois componentes os quais representam respetivamente os pedidos a registar e os pedidos submetidos.

Os pedidos a registar, são todos os que foram criados pelo cliente, mas ainda não foram submetidos, o que permite ao cliente, caso se engane nalgum dos pedidos, o possa eliminar e assim corrigir o pedido antes de ser enviado para o servidor, conforme mostrado na ilustração 15.

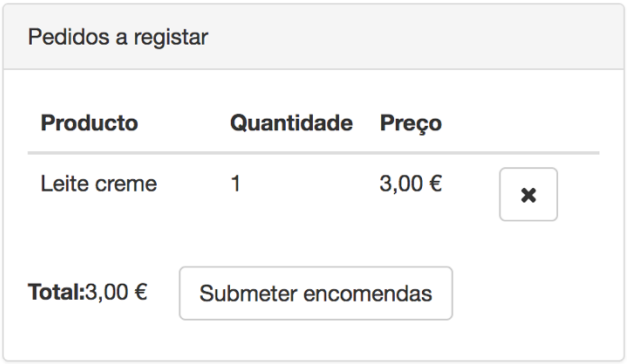


Ilustração 15: Formulário de produtos a registar

Os pedidos submetidos, mostram os artigos agrupados por pedido onde cada grupo pode ser expandido através do *icon* com sinal mais, que passa para o sinal menos quando fica expandido. Cada pedido mostra o somatório dos artigos, a quantidade e também o estado atual do pedido como pode ser visto na ilustração 16.

Ilustração 16:Componente de pedidos submetidos

Este é um separador acessível apenas a clientes.

5.4 Marcação de restaurantes nas proximidades

Na criação da componente da página *home,* surgiu a necessidade de mostrar ao utilizador os restaurantes próximos da zona pesquisada.

Para a realização dessa pesquisa, tornou-se necessário otimizar o registo das coordenadas GPS do restaurante, tendo em conta que não é possível fazer *queries* de comparação combinada por aproximação a partir do *Datastore.*

Por combinada, entenda-se a junção de mais de uma condição, que seja unida por um operador lógico como *and* ou *or*. Houve a necessidade de utilizar um algoritmo que permitisse ultrapassar esta limitação.

Para isso foi utilizada uma biblioteca com a implementação do algoritmo de *geohash.* Este algoritmo permite aumentar respetivamente a precisão do local a pesquisar em 32 vezes por caracter. O primeiro carater representará uma das 32 áreas da divisão, o segundo uma das 32 dentro da primeira e assim sucessivamente, quer isto dizer que quanto maior for a precisão, maior será a cadeia de carateres.

Na tabela 5 podem ser vistos os 32 caracteres que constituem a cadeia possível no *geohash*.

Tabela 4: Base 32 para geohashing

Decimal	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Base 32	0	1	2	3	4	5	6	7	8	9	b	c	d	e	f	g
Decimal	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
Base 32	h	j	k	m	n	p	q	r	s	t	u	v	w	x	y	z

Para poder reduzir ainda a possibilidade de haver uma vizinhança próxima, não ser contabilizada, foram considerados 9 *geohashes* como o exemplo abaixo na tabela 17, onde a vermelho se encontra o *geohash* a considerar.

Este *geohash,* pode ser obtido pela ajuda da biblioteca *geohash-java,* onde é possível através da latitude, longitude e número de carateres, saber o *geohash* correspondente.

Como forma de otimizar a velocidade de pesquisa foi adicionado este campo diretamente na entidade.

Geohash vizinho	Geohash vizinho	Geohash vizinho
Geohash vizinho	Geohash a considerar	Geohash vizinho
Geohash vizinho	Geohash vizinho	Geohash vizinho

Ilustração 17: Tabela de vizinhança

5.5 Módulos e build

Esta foi uma aplicação criada desde o início pensada tendo em conta a modularidade. Foi por essa razão que foram criados três módulos iniciais.

São estes módulos o de *frontend*, o qual foi nomeado de *web-client*, o de *backend* o qual foi nomeado de *web-server* e o módulo partilhado entre estes, denominado *shared*.

A razão para esta iniciativa foi manter um baixo vínculo entre os módulos, sendo possível num futuro próximo a substituição de cada um deles criando implementações diferentes das atuais, além disso torna-se mais fácil testar estes módulos, dado que basta apenas simular o comportamento dos módulos adjacentes.

Para fazer a integração entre estes módulos, assim como acelerar o processo e *debug* foi necessário automatizar alguns processos. Para isso foram criadas *tasks gradle*.

Para a execução de uma *build GWT,* houve a necessidade de adicionar e configurar o *plugin GWT* do *gradle* no módulo *web-client*, assim como o *plugin* do *appengine* e *war* o primeiro para gerir o modulo para a plataforma da Google e o segundo para gerir a criação do *package web*, padrão do *JAVA-EE*.

Além destes *plugins*, foram também adicionados os associados à linguagem *java* e *groovy* para que fosse possível compilar o código fonte.

As *tasks* que permitem gerir o módulo *GWT*, importantes de mencionar no contexto da implementação, são abaixo descritas:

- *Clean* – esta é uma *task* global a todos os módulos *gradle,* que possibilita por padrão, a remoção da pasta *build* do módulo em causa;

- *CompileGwt* – como o próprio nome indica esta *task* permite compilar o código *GWT*, que fará transformar o código *java* em código *javascript*. Este novo código será gerado no caminho relativo à pasta de *build* do módulo em *build/gwt/out/index*;

- *GwtSuperDev* – esta *task* dá acesso a um *url*, que permite compilar a aplicação no browser em modo de *debug* após clicar no link criado [20], o qual fará aparecer no browser a mensagem apresentada na ilustração 18;

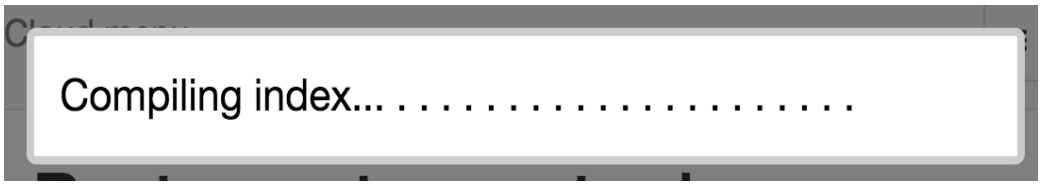


Ilustração 18:Mensagem de compilação no browser para o modo de debug

Este modo permite ver o código *java* feito inicialmente pelo programador, além de que é possível efetuar pausas no *browser* nesse código *java,* a fim de identificar problemas no funcionamento da aplicação.

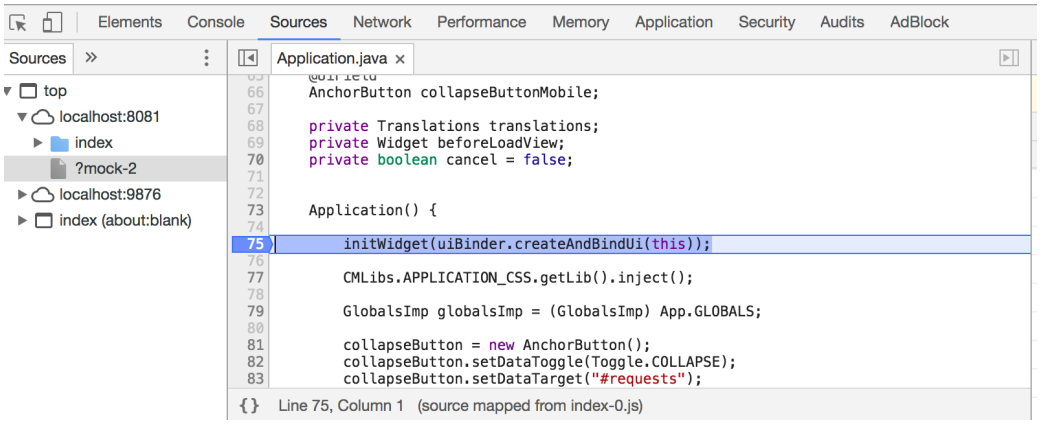


Ilustração 19: Ferramentas do programador do Chrome em modo de debug pausado na linha 75 da class Aplication

- *AppengineRun – task* que inicializa o servidor local que lê na pasta *exploded-app* na pasta *build* do módulo do servidor.

Aos *build scripts do gradle,* é ainda possível acrescentar *tasks* com recurso à linguagem *groovy*, tal como foi dito no capítulo 5.2., nessas *tasks* podem ser configuradas dependências, o que faz com que essas dependências sejam executadas antes da *task* em causa.

Foram criadas as seguintes *tasks* com as características mencionadas abaixo:

- *CopyClientToExploded* – é uma *task* acrescentada na *build* do módulo *web-server* que copia o conteúdo da pasta *build/gwt/out/index* do módulo *web-client* para a pasta *build/exploded-app/index* do módulo *web-server,* o que permite após o servidor estar online poder substituir o conteúdo estático gerado pelo módulo *web-client*;

- *CopyClientToWebapp* – é uma *task* acrescentada na *build* do módulo *web-server* que copia o conteúdo da pasta *build/gwt/out/index* do módulo *web-client* para a pasta *main/src/webapp* do módulo *web-server,* o que permite acrescentar à pasta padrão dos recursos estáticos do *web-server* os recursos gerados pelo módulo *web-client*;

- *CleanClientToExplode* – task acrescentada ao módulo web-server que limpa os recursos gerados na build antes de executar o *copyClientToExploded;*

- *CopyPublic* – Esta *task*, foi acrescentada na *build* do módulo *web-client* e copia os recursos estáticos para a pasta build/gwt/out/índex do módulo;

- *CopyPublicAndCgwt – Task* que acrescenta ao módulo *web-client* a funcionalidade de compilar o código java do módulo *web-client* e copia os recursos estáticos da pasta src/main/resources/public para *build/gwt/out/índex* do mesmo módulo.

Após o desenvolvimento criado é importante validar a aplicação tema que será abordado no capítulo seguinte.

6 Verificação e validação

Neste capítulo, será descrito como a aplicação foi testada, a forma como foram feitos os testes e a sua previsão de evolução ao longo do tempo.

6.1 Testes de interface de utilizador

Este tipo de testes apesar de poderem ser automatizados com *Selenium* por exemplo, a opção do autor foi efetuá-los de forma manual.

Ainda assim, para poderem ser feitos testes de uma forma mais célere, foram efetuados maioritariamente no servidor de desenvolvimento local, navegando pelas páginas e componentes da mesma forma que será feito pelos utilizadores finais.

Como a autenticação é feita de forma externa à aplicação, houve a necessidade de simular os utilizadores, de forma a acelerar o processo de testes.

Para cumprir esse objetivo na injeção de dependências, injetou-se um objeto diferente do que terá lugar no servidor de produção. A implementação do teste assenta em criar um ficheiro que terá as propriedades do utilizador, uma das propriedades será o identificador do utilizador, que assim pode ser enviado no URL, para que seja identificado o utilizador pretendido.

Um exemplo de URL seria *http://localhost:8081/?mock-2,* o qual indica que o utilizador seria o 2 do mapa de utilizadores de testes.

6.1.1 Testes em várias dimensões de ecrã

Tendo em conta, que um dos principais cuidados no desenvolvimento foi a possibilidade de adaptação da interface a várias dimensões de ecrãs, houve a necessidade de fazer este teste.

Trata-se também de um teste manual, contudo, com a ajuda das ferramentas de programador da *Google Chrome* foi possível testar apenas no computador.

Como foram definidos via CSS alguns *media queries* e havendo alguns já definidos no CSS do *bootstrap*, durante o desenvolvimento, estes podem ser observados no *Google Chrome* e alternando entre eles, podemos observar o comportamento para cada uma das larguras definidas na aplicação como pode ser visto na ilustração 20.

Ilustração 20: Visualização do comportamento da aplicação com diferentes larguras

6.2 Testes e integração contínua

Os testes são o tipo de ferramenta que nos garante a uma maior qualidade na aplicação, mas não a perfeição.

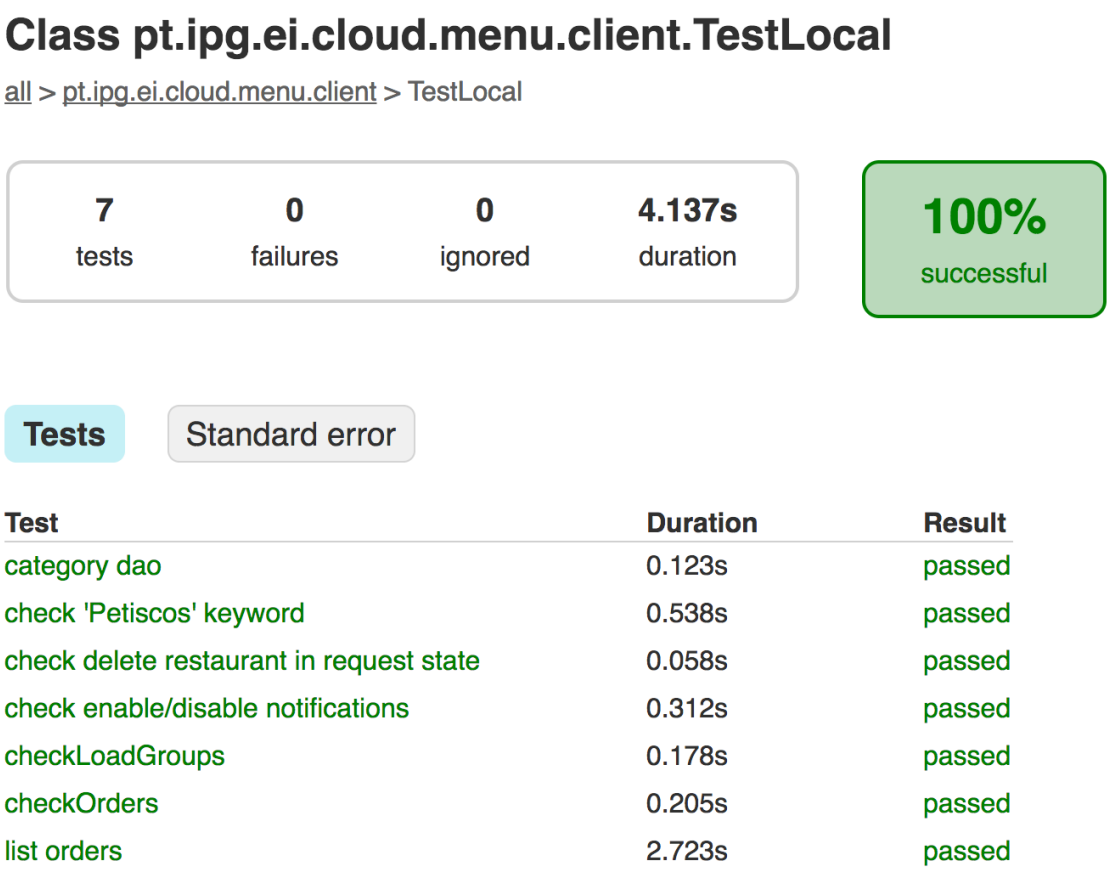
Após efetuar um teste, conseguimos validar, se o que está a ser realizado é realmente o que pretendemos, de qualquer forma, tal como no desenvolvimento em si é possível realizar testes que não têm uma abrangência tão significativa como deveria ou a consepção da aplicação não tenha sido bem feita, o que leva a uma falha de ambos, tanto dos testes como da aplicação.

Estes testes, devem ser executados de forma automática antes de fazer algum desenvolvimento, desta forma diminui a possibilidade de criar regressões na aplicação.

6.2.1 Testes unitários

Os testes unitários dão-nos uma visão unitária do desenvolvimento de um componente de código. No caso particular desta aplicação, onde foram utilizadas linguagens com o paradigma de orientação a objetos, é testado normalmente uma *class* ou um método por teste.

Estes testes por padrão ficam num local diferente da pasta de desenvolvimento, uma vez que vão ter dependências que não são necessárias na aplicação, mas sim apenas para os testes. Neste caso será *src/test* seguido da linguagem de programação correspondente, *java* ou *groovy,* onde só é considerado validado, desde que todos os testes estejam no estado *passed* como pode ser visto na ilustração 21.



Class pt.ipg.ei.cloud.menu.client.TestLocal

all > pt.ipg.ei.cloud.menu.client > TestLocal

7 tests	0 failures	0 ignored	4.137s duration	100% successful

Tests | Standard error

Test	Duration	Result
category dao	0.123s	passed
check 'Petiscos' keyword	0.538s	passed
check delete restaurant in request state	0.058s	passed
check enable/disable notifications	0.312s	passed
checkLoadGroups	0.178s	passed
checkOrders	0.205s	passed
list orders	2.723s	passed

Ilustração 21: Testes unitários de backend no módulo web-server

6.2.2 Testes de integração

Os testes de integração, visam testar a aplicação no seu todo ou na integração dos seus módulos. No caso da aplicação *cloud-restaurant* foram efetuados para testar os *web-services*, tentando simular os fluxos de dados que os utilizadores poderiam fazer na aplicação.

Também estes testes foram efetuados recorrendo à *framework TestNG* para a sua execução e configuração, mapeando listas de dados para as chamadas aos *webmethods* que se pretendia testar. Este teste foi definido no módulo *web-client*.

Para se poder executar este teste, o servidor local tem que estar ligado e para que a *build* sem parâmetros não corra os testes de integração, foram habilitados só quando passado o parâmetro *testMode=integration*. Os resultados dos testes podem ser vistos no *IDE* ou no relatório gerado na pasta *build/reports/tests/test/index.html* do módulo.

O resultado dos testes de integração pode ser visto em anexo A3.

7 Conclusões

Na elaboração deste trabalho foi possível incrementar o conhecimento, tanto a nível técnico como funcional.

Tecnicamente, foi possível perceber na prática as principais diferenças entre uma aplicação *web* tradicional e uma *single page application* e as suas implicações quando se usa a renderização no servidor ou no cliente. Foi também possível perceber as vantagens de uma arquitetura de micro serviços e as vantagens da escalabilidade horizontal entre outras coisas, tal como o enriquecimento de conhecimento a nível das linguagens de programação utilizadas e as ferramentas de *build*.

A nível funcional, foi possível perceber de que forma se podem complementar aplicações do tipo diretórios de restaurantes, podendo assim oferecer um serviço diretamente aos restaurantes, conjugando-o com o serviço dos clientes do restaurante.

7.1 Trabalho futuro

Apesar de funcionalmente esta aplicação estar mais focada na interação interna dos restaurantes, a parte do diretório de restaurante pode torná-la numa aplicação mais completa.

Ainda no contexto da interação dentro do restaurante, no futuro é importante manter o desenvolvimento de algumas funcionalidades.

- Histórico de pratos de um cliente;
- Histórico de transição de estados da encomenda;
- Promoções, por exemplo após dez refeições ter direito a um desconto igual ao consumo diário mais barato;
- *Feedback* do cliente, comentários em relação aos produtos, como forma de os avaliar;
- Listar restaurantes dos quais o utilizador é funcionário.

8 Bibliografia

[1] L. Mello da Silveira, M. Petrini e A. C. M. Zanardo dos Santos, "REGE Revista de Gestão," 19 10 2017. [Online]. Available: https://www.revistas.usp.br/rege/article/view/129033. [Acedido em 19 10 2017].

[2] I. OpenTable, "Open Table," [Online]. Available: https://www.opentable.com/.

[3] L. F. SAS, "The Fork," [Online]. Available: https://www.thefork.pt/.

[4] A. Silva e C. Videira, UML Metodologias e ferramentas case, vol. 2, Centro Atlântico.

[5] JGraph, "JGraphX," [Online]. Available: https://github.com/jgraph/jgraphx/blob/master/README.md.

[6] S. Adolph e P. Bramble, Patterns for Effective Use Cases, Boston: Addison-Wesley Longman Publishing Co., 2002.

[7] D. Allen e S. White, "Asciidoctor gradle plugin," 6 10 2016. [Online]. Available: http://asciidoctor.org/docs/asciidoctor-gradle-plugin.

[8] "PlantUML," 12 5 2016. [Online]. Available: http://plantuml.com.

[9] S. Chacon e B. Straub, Pro GIT, Second ed., Apress, 2014.

[10] D. Beaumont, "How to explain vertical and horizontal scaling in the cloud," [Online]. Available: https://www.ibm.com/blogs/cloud-computing/2014/04/explain-vertical-horizontal-scaling-cloud/. [Acedido em 25 7 2016].

[11] Google, "Cloud Storage Options | Google Cloud Platform," [Online]. Available: https://cloud.google.com/storage/.

[12] Google, "Google+ API | Google+ Platform for Web | Google Developers," [Online]. Available: https://developers.google.com/+/web/api/rest/.

[13] O. Corporation, “Jersey,” [Online]. Available: https://jersey.github.io/.

[14] Google, “Guice,” [Online]. Available: https://github.com/google/guice.

[15] SmartBear, “Swager,” [Online]. Available: https://swagger.io/.

[16] Google, “GWT,” [Online]. Available: http://www.gwtproject.org/.

[17] S. Jardine, “gwtbootstrap3,” [Online]. Available: https://github.com/gwtbootstrap3/gwtbootstrap3.

[18] C. Meier, “Resty GWT,” [Online]. Available: https://resty-gwt.github.io/.

[19] C. Beust, “TestNG,” [Online]. Available: http://testng.org/doc/.

[20] Google, [Online]. Available: http://www.gwtproject.org/articles/superdevmode.html. [Acedido em 20 2 2016].

[21] Gradle Inc., “Gradle build tool,” Gradle Inc., [Online]. Available: https://gradle.org. [Acedido em 03 02 2016].

[22] Google, “Google Web Toolkit,” Google, [Online]. Available: http://www.gwtproject.org/. [Acedido em 04 05 2016].

Anexos

A 1. Dicionário de dados

CategoryEntity

Entidade que representa as categorias de produtos

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
name	String	Nome da categoria
restaurant	Ref	Referência do restaurante
products	List	Lista de referências da entidade ProductEntity
restaurante	Key	Referencia da entidade RestaurantEntity

ProductEntity

Entidade que representa um produto

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
name	String	Nome do produto
price	Int	Preco atual do produto em cêntimos
image	String	Caminho da imagem
category	Key	Referência para registo da entidade CategoryEntity

OrderEntity

Esta é a entidade que representa os pedidos

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
state	OrderState	Estado do tipo enum declarado em OrderState
syncGroup	Key	Referência para a entidade SyncGroupEntity
client	Key	Referencia para a entidade CleintEntity
products	Map	Mapa de referências cuja chave é uma referência à entidade ProductEntity e o valor um valor inteiro com a quantidade do produto

SyncGroupEntity

Entidade que representa um grupo de pedidos que só transitarão para o estado *Ready* assim que todas os pedidos do grupo estiverem no estado *Sync*

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
orderEntities	List	Lista de referências da entidade OrderEntity

TableEntity

Entidade que representa uma mesa no restaurante

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
name	String	Nome da mesa
restaurant	Key	Referência à entidade RestaurantEntity
clients	Set	Set de referências da entidade ClientEntity

ClientEntity

Entidade que representa um cliente genérico

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
table	Key	Referência à entidade TableEntity

TempClientEntity

Entidade que representa um cliente temporário, ou seja, um cliente sem a aplicação cloud-restaurant.

Esta entidade herda da entidade ClientEntity.

Nome do campo	Tipo de dados	Descrição
name	String	Nome do cliente

AppClientEntity

Representação de um cliente registado na aplicação.

Esta entidade de ClientEntity.

Nome do campo	Tipo de dados	Descrição
userAppEntity	Key	Referencia à entidade UserAppEntity

UserAppEntity

Representação de um utilizador da aplicação

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
googleUser	Key	Referência à entidade GoogleUserEntity
displayName	String	Nome que será mostrado na aplicação
email	String	*E-mail* do utilizador
admin	Boolean	Atribuição de direitos de administrador ao utilizador
myRestaurant	Key	Lista de restaurantes registados pelo utilizador

GoogleUserEntity

Representação da entidade com a informação que é capturada no *GooglePlus* API

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
displayName	String	Nome que será mostrado na aplicação
email	String	*E-mail* do utilizador

PairTableEntity

Representa o emparelhamento de um clinte

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
table	Key	Referência à entidade TableEntity
state	enum	Estado do emparelhamento com as constantes do enum RestaurantStatePair
user	Key	Referência da entidade UserAppEntity que representa o utilizador a emparelhar
token	String	String aleatória do tipo UUID para confirmação visual de correspondência
tempQt	String	Quantidade de clientes temporários

RestaurantStaffMemberEntity

Representação de um funcionário do restaurante

Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade
userAppKey	Key	Referência da entidade UserAppEntity
restaurantKey	Key	Referência da entidade RestaurantEntity
active	Boolean	Estado activo do funcionário

KitStaffEntity

Representação de um funcionário de cozinha.

Esta entidade não tem dados, é apenas uma subclasse de RestaurantStaffMemberEntity.

WaiterEntity

Representação de um funcionário de mesa.

Esta entidade não tem dados, é apenas uma subclasse de RestaurantStaffMemberEntity.

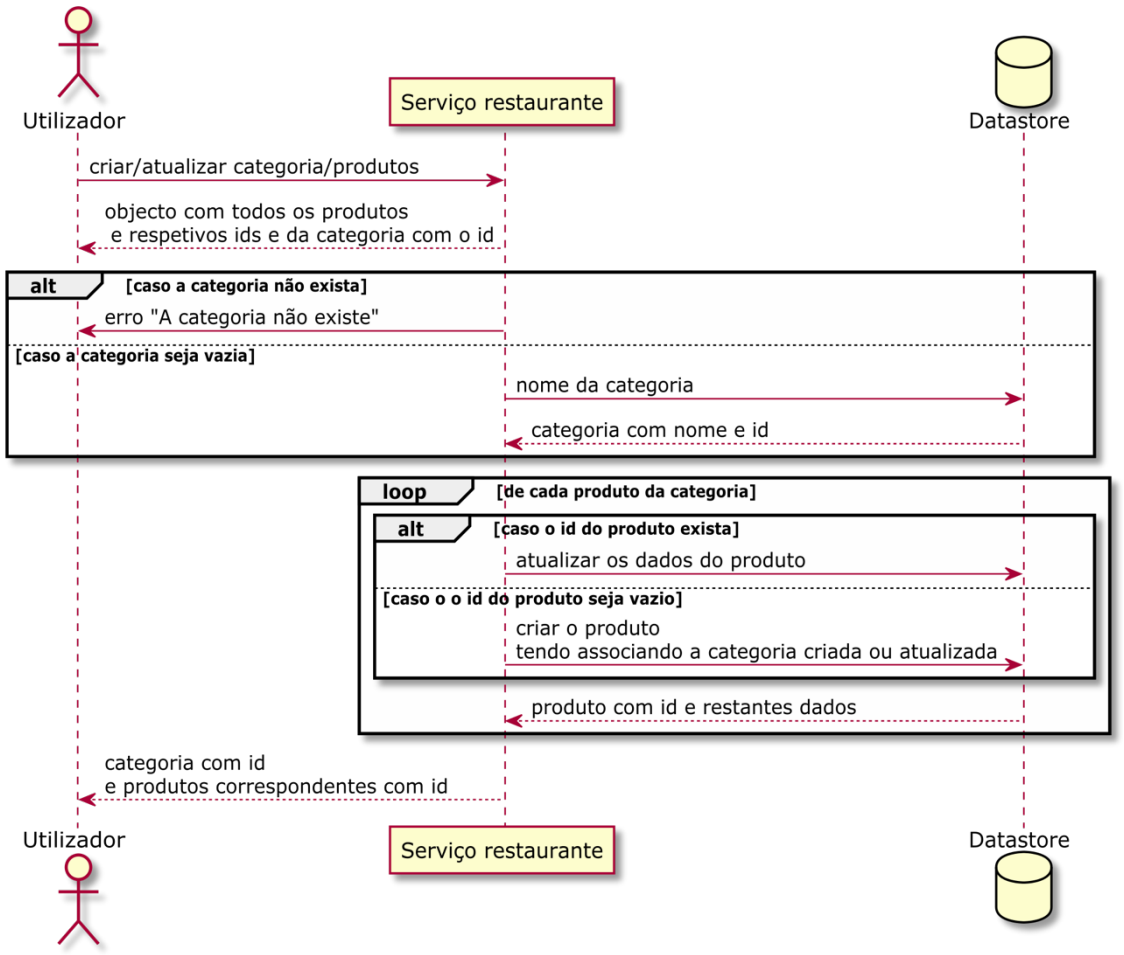
RestaurantUserTypesEntity

Representação do utilizador no contexto do restaurante

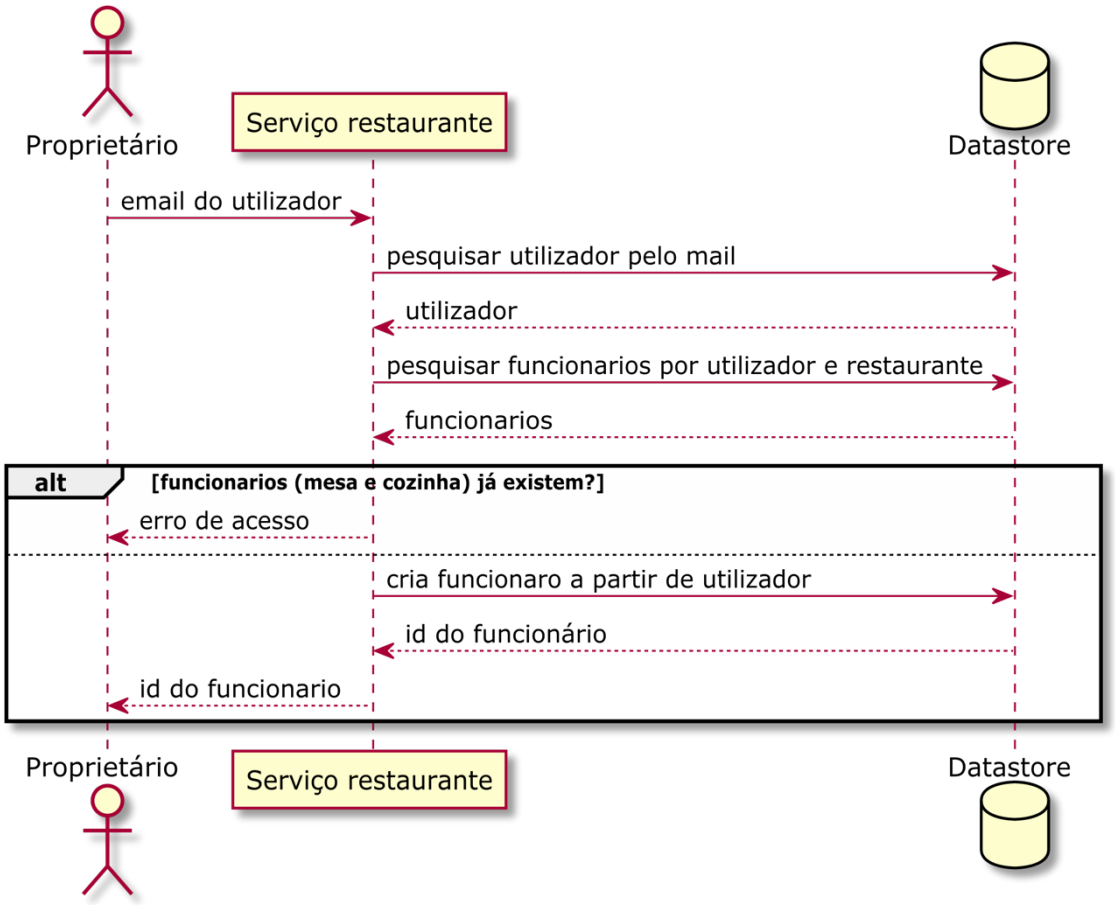
Nome do campo	Tipo de dados	Descrição
id	Long	Identificador da entidade que têm correspondência com o identificador do restaurante
userAppEntityKey	Key	Referência da entidade UserAppEntity
type	Int	Tipo de utilizador do restaurante com a correspondência 0: não existe ligação ao restaurante 1: Funcionário de mesa 2: Cliente 3: Funcionário de cozinha
restaurantUserId	long	Corresponde ao identificador de uma das entidades WaiterEntity, AppClientEntity ou KitStaffEntity tendo em conta o conteúdo do campo type

A 2. Diagramas de sequência

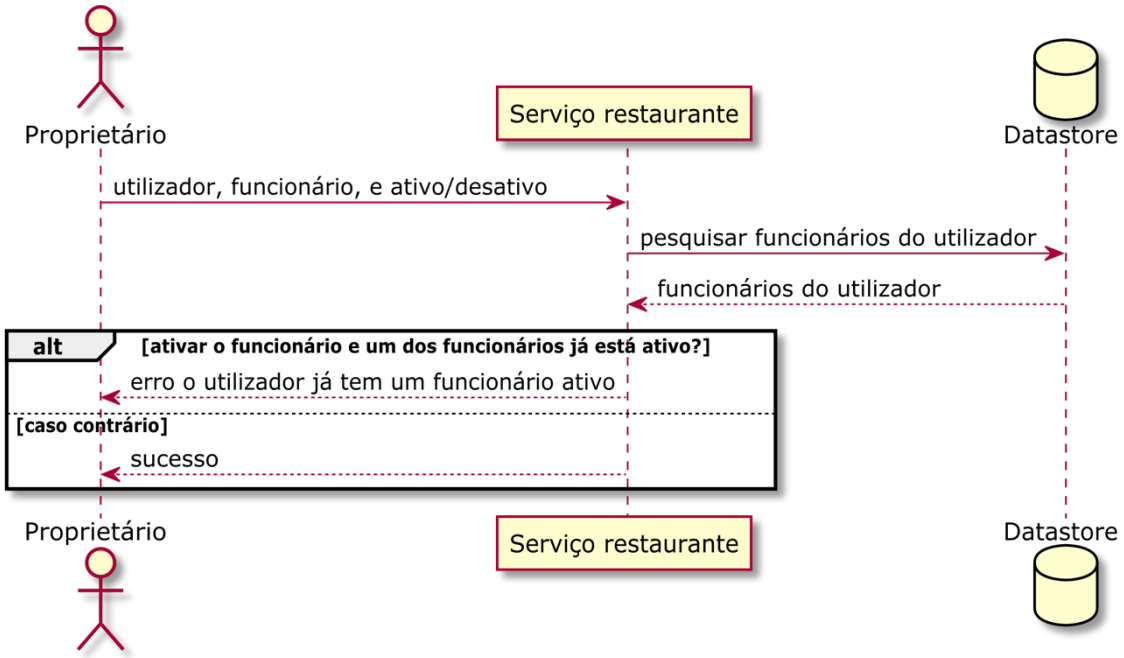
Criação/atualização categoria



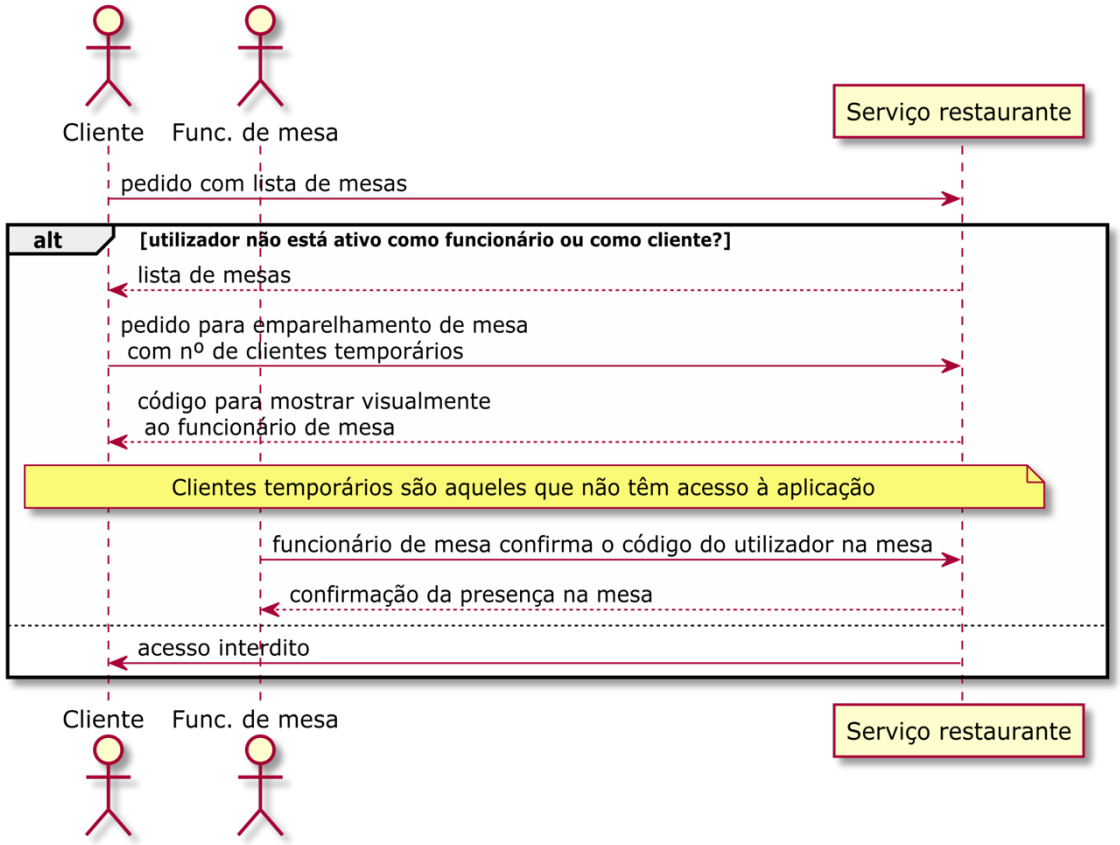
Criação de um funcionário



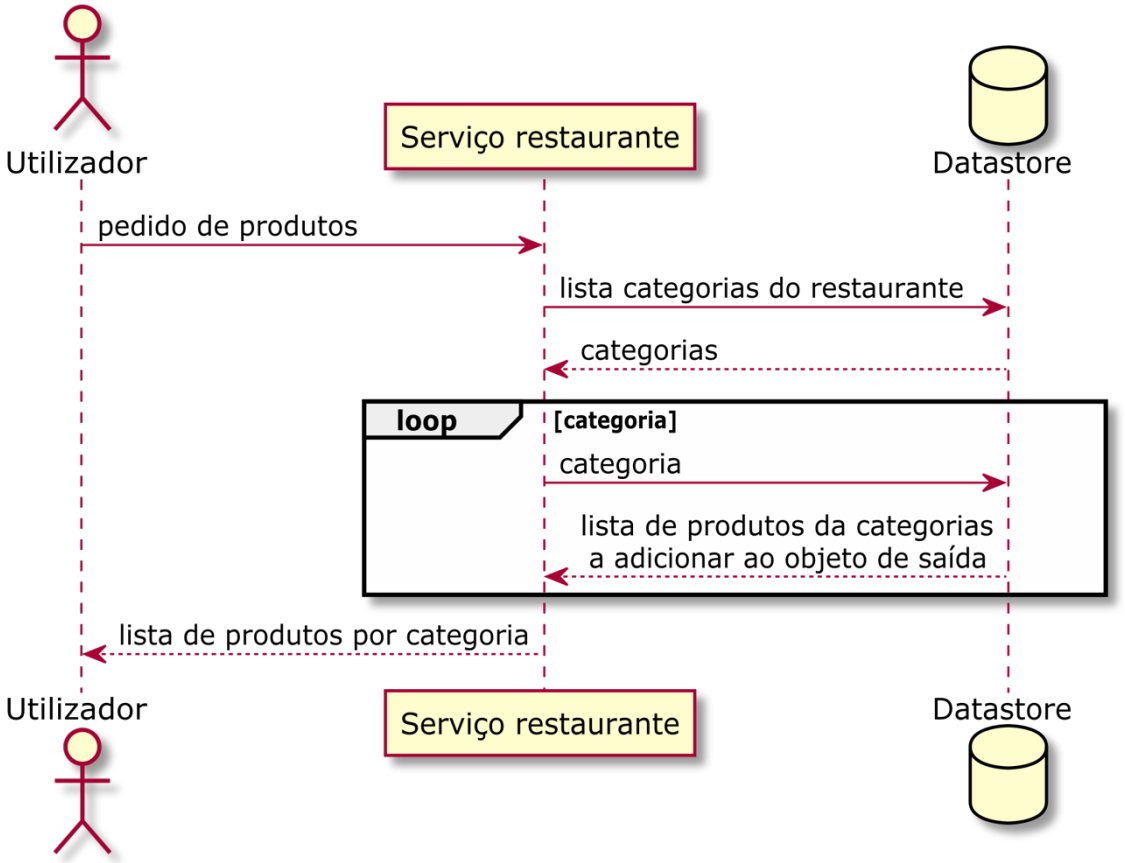
Ativar/Desativar funcionário



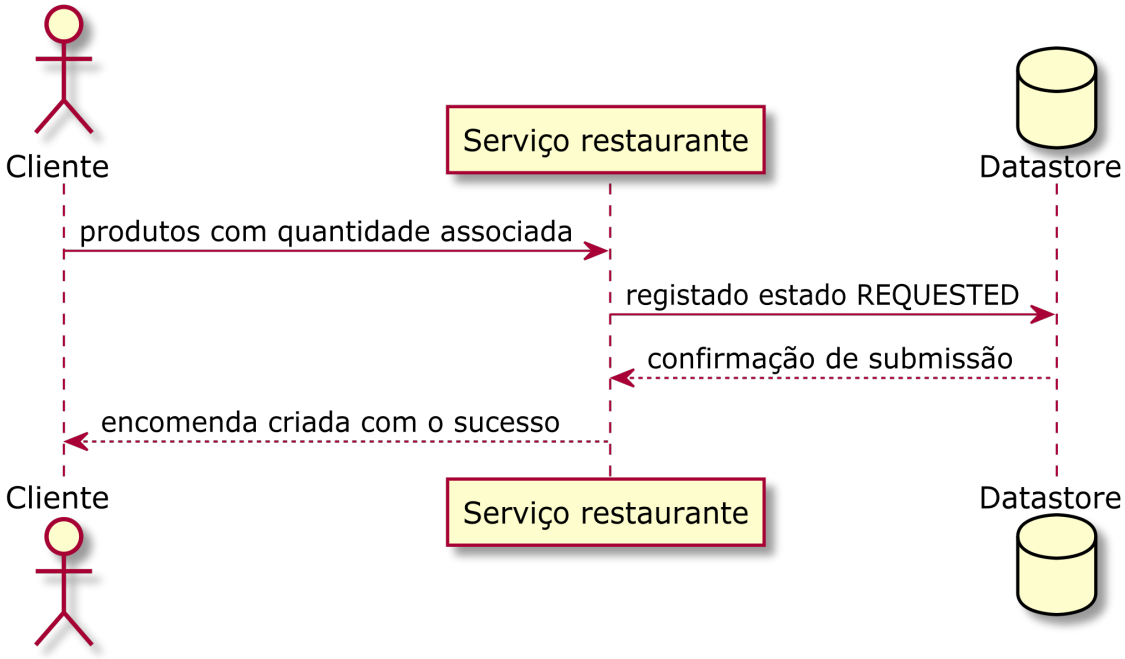
Confirmação/emparelhamento do cliente



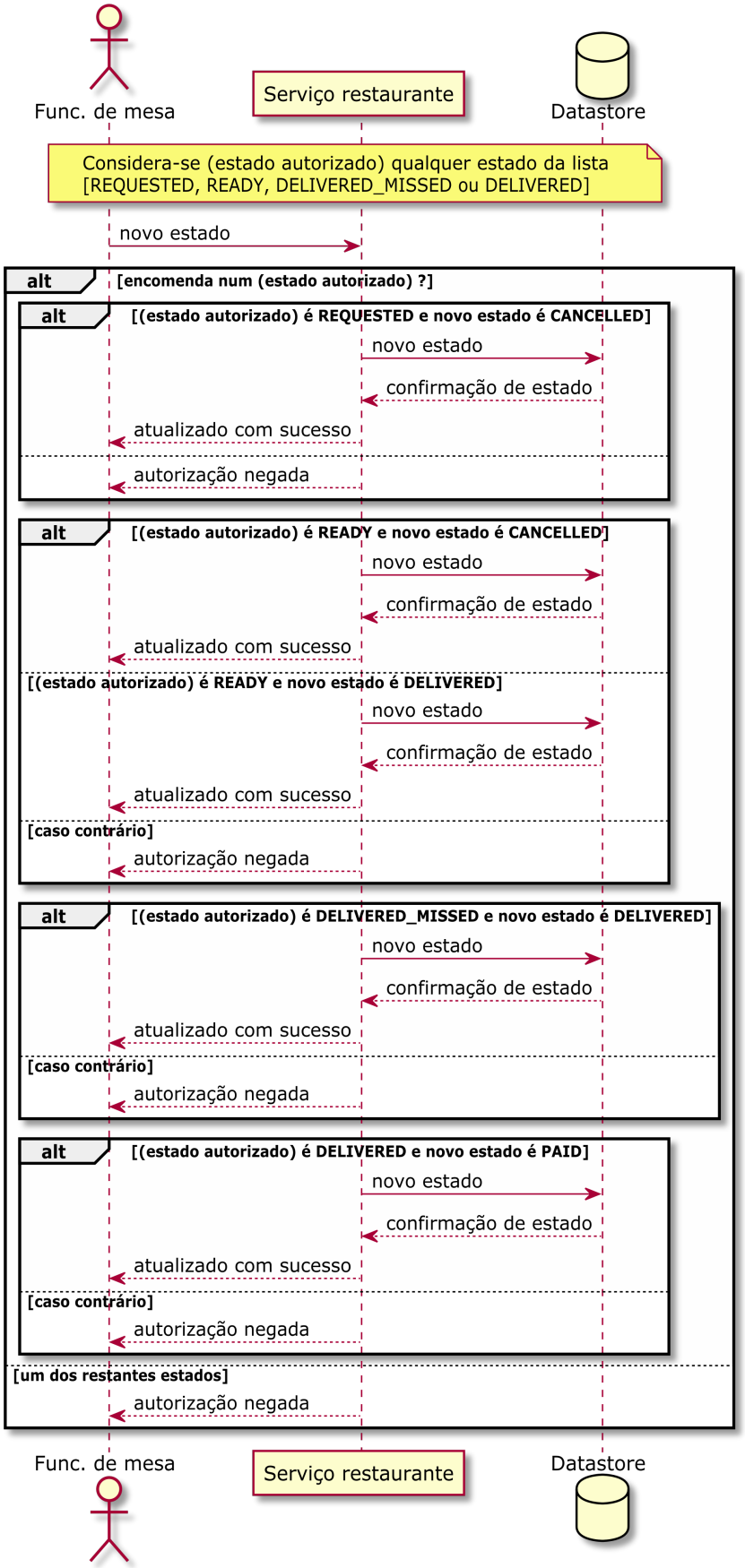
Listar categorias e produtos



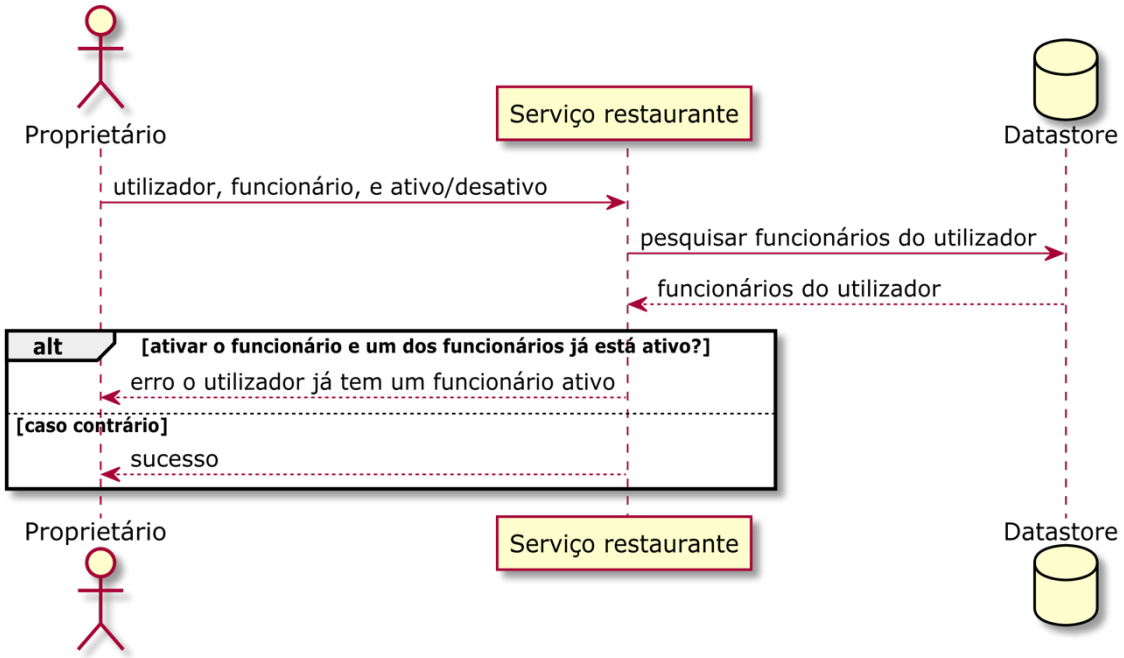
Submissão de encomenda



Atualização da encomenda pelo funcionário de mesa



Atualização de encomenda pelo funcionário de cozinha



A 3. Resultado dos testes de integração

Test	Duration	Result
addProducts[0](restaurant[id=4636090778517504, name=Petisco])	0.388s	passed
addProducts[0](restaurant[id=5761990685360128, name=Sardinha])	0.388s	passed
addStaff[0](restaurant[id=4636090778517504, name=Petisco])	0.102s	passed
addStaff[0](restaurant[id=5761990685360128, name=Sardinha])	0.105s	passed
addTables[0](restaurant[id=4636090778517504, name=Petisco])	0.072s	passed
addTables[0](restaurant[id=5761990685360128, name=Sardinha])	0.072s	passed
confirmPairingByWaiters[0](restaurant[id=4636090778517504, name=Petisco], user[googleId=11111111111111111112, displayName=Mock User 2, email=user2@gmail.com])	1.431s	passed
confirmPairingByWaiters[0](restaurant[id=5761990685360128, name=Sardinha], user[googleId=11111111111111111114, displayName=Mock User 4, email=user4@gmail.com])	1.426s	passed
nearMe[0](40.5385778, -7.2801131, 5)	0.044s	passed
nearMe[0](40.5385778, -7.2801131, 6)	0.034s	passed
orders[0](table[name=1], 1)	0.718s	passed
orders[1](table[name=1], 2)	0.454s	passed
orders[2](table[name=1], 3)	0.413s	passed
orders[3](table[name=1], 1)	0.457s	passed
orders[4](table[name=1], 2)	0.466s	passed
orders[5](table[name=1], 3)	0.446s	passed
pairRequest[0](restaurant[id=4636090778517504, name=Petisco], table[name=1], user[googleId=11111111111111111117, displayName=Mock User 7, email=user7@gmail.com])	0.103s	passed
pairRequest[0](restaurant[id=4636090778517504, name=Petisco], table[name=1], user[googleId=11111111111111111118, displayName=Mock User 8, email=user8@gmail.com])	0.112s	passed
pairRequest[0](restaurant[id=5761990685360128, name=Sardinha], table[name=1], user[googleId=11111111111111111113, displayName=Mock User 3, email=user3@gmail.com])	0.091s	passed
pairRequest[0](restaurant[id=5761990685360128, name=Sardinha], table[name=1], user[googleId=11111111111111111115, displayName=Mock User 5, email=user5@gmail.com])	0.091s	passed

Test	Duration	Result
restaurantConfirm[0](restaurant[id=null, name=Petisco])	0.320s	passed
restaurantConfirm[0](restaurant[id=null, name=Sardinha])	0.429s	passed
restaurantRegister[0](restaurant[id=null, name=Petisco])	0.592s	passed
restaurantRegister[0](restaurant[id=null, name=Sardinha])	0.592s	passed
tablesCheck[0](restaurant[id=4636090778517504, name=Petisco])	0.033s	passed
tablesCheck[0](restaurant[id=5761990685360128, name=Sardinha])	0.033s	passed
testUser[0](user[googleId=111111111111111111110, displayName=Mock User 10, email=user10@gmail.com])	0.170s	passed
testUser[0](user[googleId=111111111111111111112, displayName=Mock User 2, email=user2@gmail.com])	0.190s	passed
testUser[0](user[googleId=111111111111111111113, displayName=Mock User 3, email=user3@gmail.com])	0.272s	passed
testUser[0](user[googleId=111111111111111111114, displayName=Mock User 4, email=user4@gmail.com])	0.198s	passed
testUser[0](user[googleId=111111111111111111115, displayName=Mock User 5, email=user5@gmail.com])	0.253s	passed
testUser[0](user[googleId=111111111111111111116, displayName=Mock User 6, email=user6@gmail.com])	0.269s	passed
testUser[0](user[googleId=111111111111111111117, displayName=Mock User 7, email=user7@gmail.com])	0.210s	passed
testUser[0](user[googleId=111111111111111111118, displayName=Mock User 8, email=user8@gmail.com])	0.196s	passed
testUser[0](user[googleId=111111111111111111119, displayName=Mock User 9, email=user9@gmail.com])	0.268s	passed
testUser[0](user[googleId=112151260953818791414, displayName=Mock Administrator, email=xico@gmail.com])	0.275s	passed
testUser[2](user[googleId=111111111111111111111, displayName=Mock User 11, email=user11@gmail.com])	0.094s	passed
testUser[2](user[googleId=111111111111111111112, displayName=Mock User 12, email=user12@gmail.com])	0.107s	passed
testUser[3](user[googleId=111111111111111111113, displayName=Mock User 13, email=user13@gmail.com])	0.091s	passed
testUser[4](user[googleId=111111111111111111114, displayName=Mock User 14, email=user14@gmail.com])	0.075s	passed

Test	Duration	Result
testUser[5](user[googleId=111111111111111111115, displayName=Mock User 15, email=user15@gmail.com])	0.072s	passed
userRestaurantWaiter[0](user[googleId=111111111111111111110, displayName=Mock User 10, email=user10@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.011s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111112, displayName=Mock User 2, email=user2@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.050s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111113, displayName=Mock User 3, email=user3@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.024s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111114, displayName=Mock User 4, email=user4@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.055s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111115, displayName=Mock User 5, email=user5@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.022s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111116, displayName=Mock User 6, email=user6@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.046s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111117, displayName=Mock User 7, email=user7@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.015s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111118, displayName=Mock User 8, email=user8@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.013s	passed
userRestaurantWaiter[0](user[googleId=11111111111111111119, displayName=Mock User 9, email=user9@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.038s	passed
userRestaurantWaiter[0](user[googleId=112151260953818791414, displayName=Mock Administrator, email=xico@gmail.com],	0.024s	passed

Test	Duration	Result
[restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])		
userRestaurantWaiter[4](user[googleId=111111111111111111111, displayName=Mock User 11, email=user11@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.002s	passed
userRestaurantWaiter[5](user[googleId=111111111111111111112, displayName=Mock User 12, email=user12@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.002s	passed
userRestaurantWaiter[5](user[googleId=111111111111111111113, displayName=Mock User 13, email=user13@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.002s	passed
userRestaurantWaiter[5](user[googleId=111111111111111111114, displayName=Mock User 14, email=user14@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.002s	passed
userRestaurantWaiter[5](user[googleId=111111111111111111115, displayName=Mock User 15, email=user15@gmail.com], [restaurant[id=4636090778517504, name=Petisco], restaurant[id=5761990685360128, name=Sardinha]])	0.002s	passed